Università degli Studi di Messina
Dipartimento di Scienze Matematiche e Informatiche
Scienze Fisiche e Scienze della Terra
Corso di Laurea Triennale in Informatica

Sistemi di certificazione digitale basati su blockchain permissioned in ambito accademico

TESI DI LAUREA DI:
BERNARDO DE PIETRO

RELATORE:
Prof. Ing. Massimo Villari

CORRELATORE:
Armando Ruggeri

Anno Accademico 2020-2021

Indice

Elenco delle figure

Introduzione

A partire dal 2022, la spesa mondiale per l'adozione di soluzioni blockchain ha raggiunto l'ammontare di 11 miliardi di dollari. Si stima che il mercato globale della tecnologia blockchain accumulerà 20 miliardi di dollari di entrate entro il 2024.
Per capire meglio l'impatto che ha avuto questa tecnologia su gli utenti di tutto il mondo basti pensare che il numero di wallets blockchain registrati nel corso del secondo trimestre del 2021 è stato di oltre 70 milioni.
La comprensione delle proprietà e funzionalità della tecnologia blockchain è comunemente associata al Bitcoin. All'interno del documento "*Bitcoin: A Peer-To-Peer Electronic Cash System*, dietro lo pseudonimo di Satoshi Nakamoto l'autore descrisse il protocollo Bitcoin, mettendo in risalto l'impiego della tecnologia peer-to-peer per lo scambio di valuta digitale: il bitcoin, per l'appunto. Detto funzionamento consiste nella possibilità di effettuare transazioni di valute digitali direttamente tra due utenti, eliminando cosi un'eventuale terza figura, come istituzioni finanziarie o governative. Proprio a tale scopo, Satoshi Nakamoto introdusse la tecnologia blockchain, con l'obiettivo di utilizzarla come "libro mastro" di tutte le transazioni avvenute.
Sebbene l'implementazione della blockchain divenne nota grazie all'utilizzo che ne venne fatto all'interno del protocollo Bitcoin, il suo nucleo emerse tra la fine degli anni Ottanta e gli inizi degli anni Novanta.

Nel 1989, Leslie Lamport sviluppò il protocollo Paxos e nel 1990 rilasciò il documento "*The Part-Time Parliament to ACM Transaction on Computer Systems*". L'elaborato prospetta un modello di consenso per il raggiungimento di un accordo all'interno di una rete di computer in cui quest'ultimi o la stessa rete non sono affidabili. Tale condizione viene attuata dal protocollo Paxos il quale stabilisce che ogni transazione, essendo non affidabile, debba avere il consenso di una determinata quantità di partecipanti alla rete prima di essere validata.

In un mondo sempre più incerto, con una varietà di utenti non affidabili, ansiosi di interferire nelle operazioni per il proprio profitto, la tecnologia blockchain crea una struttura altamente resistente alle manomissioni, che può rendere ogni tipo di transazione sicura e verificabile. Nonostante le varie polemiche, il modello blockchain, attraverso il consenso digitale distribuito, è una tecnologia che potrà (in piccola parte lo sta già facendo) ridisegnare diversi settori, tra i quali quello economico, sanitario e molti altri.

CAPITOLO 1

Stato dell'arte

In questo capitolo verrà introdotta dettagliatamente dal funzionamento di base fino alle sue proprietà e peculiarità la tecnologia blockchain. La blockchain utilizza un insieme di diverse tecnologie e concetti; nello specifico verranno descritti i sistemi decentralizzati, le architetture peer-to-peer e la crittografia.

1.1 Sistemi decentralizzati

Il concetto di decentralizzazione spesso genera molta confusione. A tal proposito può essere presa in considerazione la definizione fornita dal NIST (National Institute of Standards and Technology) che presenta una rete decentralizza come: "Una configurazione di rete in cui sono presenti più autorità che fungono da hub centralizzato per una sottosezione di partecipanti. Poiché alcuni partecipanti si trovano dietro un hub centralizzato, la perdita di tale hub impedirà a tali partecipanti di comunicare." Questo concetto è stato spiegato da Vitalik Buterin, inventore di Ethereum, nell'articolo *"The Meaning of Decentralization"* che etichetta la parola "Decentralizzazione" come uno dei termini informatici definiti peggio.

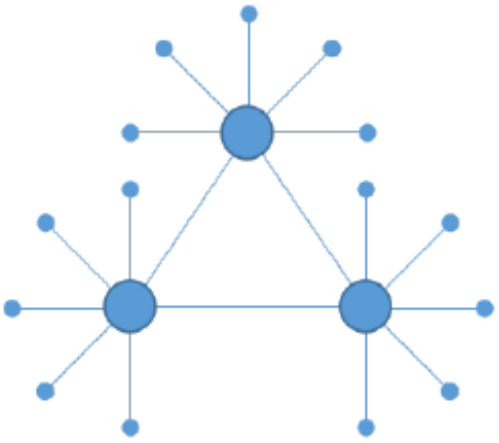

Figura 1.1: Rete decentralizzata

Quando si parla di decentralizzazione del software, si presuppone che vi siano tre assi separati di centralizzazione/decentralizzazione. Gli assi sono i seguenti:

- (De)centralizzazione architetturale: da quanti computer fisici è composto un sistema? Quanti di quei computer può tollerare che si guastino in qualsiasi momento?
- (De)centralizzazione politica: quanti individui o organizzazioni controllano in ultima analisi i computer di cui è composto il sistema?
- (De)centralizzazione logica: l'interfaccia e le strutture dati che il sistema presenta e mantiene assomigliano più a un singolo oggetto o uno sciame senza forma?

Provando a esaminare alcuni esempi è possibile rispondere alle domande poste sopra:

- Le società tradizionali sono centralizzate politicamente (un CEO), centralizzate architetturalmente (una sede centrale) e centralizzate logicamente (non si possono dividere a metà);
- Le lingue sono logicamente decentralizzate. Ad esempio, l'inglese parlato tra Alice e Bob e l'inglese parlato tra Charlie e David non deve essere necessaria-

mente uguale. Non è richiesta un'infrastruttura centralizzata per la lingua e le regole dalla grammatica non vengono controllate da una singola entità;

- BitTorren è logicamente decentralizzato in modo simile all'inglese. Le reti di distribuzione dei contenuti sono simili, ma vengono controllate da un'unica società;

- Le blockchain sono decentralizzate politicamente (nessuno le controlla) e architetturalmente (nessun punto centrale nella rete) ma logicamente centralizzate (c'è uno stato comunemente concordato e il sistema appare come un singolo computer).

Dunque è ora possibile presentare tre ragioni per le quali la decentralizzazione del software può essere utile:

- Tolleranza agli errori: è meno probabile che i sistemi decentralizzati si guastino accidentalmente perché si basano su molti componenti separati;

- Resistenza agli attacchi: i sistemi decentralizzati sono più costosi da attaccare, distruggere o manipolare perché non hanno punti centrali;

- Resistenza alle collusioni: è molto difficile accordarsi tra entità per avvantaggiarsi rispetto ad altre.

Per quanto riguarda la tolleranza agli errori il suo principio si basa sul fatto che esistono basse probabilità che cinque computer si possano guastare contemporaneamente. Tuttavia, tale probabilità potrebbe diventare una certezza se venisse trovata una falla nel sistema. Una blockchain può evitare che si verifichi una situazione del genere? Ovviamente no, sebbene all'interno di una blockchain esistano diversi meccanismi per abbassare di molto le possibilità di errori.
Nel caso della resistenza agli attacchi, la decentralizzazione si rivela alquanto fondamentale. Per fare un esempio concreto, si ipotizza che una società abbia dieci diverse sedi. Se un attaccante volesse distruggere la società dovrebbe quindi distruggere tutte

quante le sedi. Al contrario, se la società possedesse un'unica sede, essa potrebbe essere distrutta molto più facilmente.
Infine, si ha la resistenza alle collusioni. Chiarire attraverso un esempio pratico il concetto di collusione sarebbe più difficoltoso che spiegarlo come "*un coordinamento di entità che non ci piace*". Un sistema decentralizzato impiega dei meccanismi tali da contrastare la collusione.

1.2 Architetture peer-to-peer

Peer-to-peer, o P2P nella sua forma abbreviata, si riferisce a reti di computer che utilizzano un'architettura distribuita. Nelle reti P2P, tutti i computer e i dispositivi che ne fanno parte sono indicati come peer, i quali condividono e scambiano carichi di lavoro. Tutti i peer sono uguali tra loro in una rete di questo genere; non vi sono peer privilegiati, né un dispositivo di amministrazione principale al centro della rete.

1.2.1 Obiettivo

L'obiettivo principale delle reti peer-to-peer è quello di condividere risorse e aiutare computer e dispositivi a lavorare in modo collaborativo, fornire servizi specifici o eseguire determinate attività. L'impiego più comune per queste reti è la condivisione su Internet: sono di fatto ideali per la condivisione di file perché consentono ai computer ad esse collegati di ricevere e inviare file simultaneamente.

1.2.2 Utilità

Le reti peer-to-peer posseggono alcune caratteristiche che le rendono molto utili:

- Sono difficili da abbattere. Anche se uno dei peer viene spento, gli altri continuano a funzionare e comunicare. Affinché una rete peer-to-peer smettesse di funzionare, sarebbe necessario chiudere tutti i suoi peer;

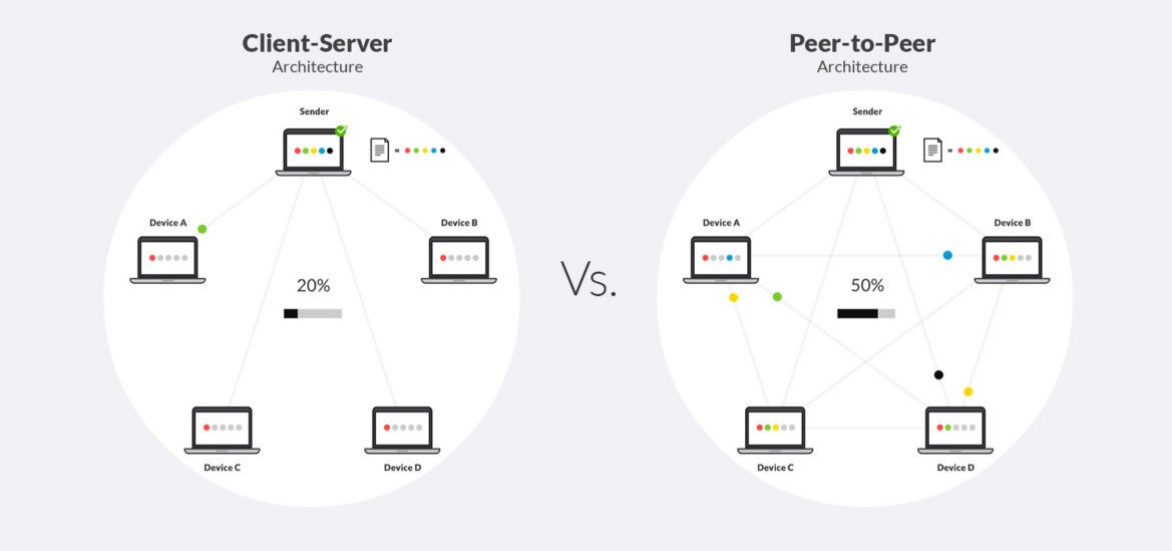

Figura 1.2: Differenza tra architetture client-server e peer-to-peer

- Sono altamente scalabili. L'aggiunta di nuovi peer è agevole in quanto non è necessario eseguire alcuna configurazione su un server centrale.

Nel caso della condivisione di file, più grande è la rete peer-to-peer più veloce sarà la condivisione. Avere lo stesso file archiviato su molti dei peer della rete significa che, ogni qual volta qualcuno dovrà scaricare il file, questo verrà scaricato da più posizioni contemporaneamente.

1.3 Crittografia

Durante la stesura di questo elaborato verrà spesso utilizzata la parola chiave "*hash*". L'hash è un elemento saliente facente parte del macro argomento della crittografia. Per tal motivo si introdurrà la crittografia in generale e, successivamente, vi si concentrerà sulla crittografia asimmetrica e sulle funzioni hash.

1.3.1 Definizione di crittografia

La crittografia si compone di una serie di metodi per occultare un messaggio in modo da non essere comprensibile a persone non autorizzate a leggerlo. La crittografia consta di alcuni elementi, tra cui:

- Sistema crittografico: realizza la funzionalità di natura crittografica, quindi è un algoritmo o un protocollo;
- Testo in chiaro: è il contenuto originale del messaggio comprensibile a chiunque;
- Testo cifrato: è l'alterazione volontaria e reversibile del messaggio originale, cioè il testo in chiaro;
- Chiave: è utilizzata dai metodi di cifratura per cifrare il testo in chiaro. La chiave è nota solo al mittente e al destinatario.

1.3.2 Crittografia asimmetrica

Esistono due principali sistemi crittografici:

- Sistema simmetrico: è il sistema che utilizza solamente delle chiavi segrete per cifrare i messaggi;
- Sistema asimmetrico: esso fornisce ad ogni attore in comunicazione due chiavi (una chiave pubblica visibile a tutti e una chiave privata).

Optando per una crittografia asimmetrica, è possibile realizzare due funzioni. Una che utilizzi la chiave segreta del mittente per *autenticare* il messaggio inviato al destinatario e una seconda funzione che si serva della chiave pubblica del destinatario per garantire la *segretezza* del messaggio.
In pratica, nei sistemi che adottano questo tipo di crittografia chiunque può utilizzare la chiave pubblica di un utente per cifrare un messaggio ma, contestualmente, lo stesso messaggio può essere decifrato solamente con la chiave privata del destinatario.
La sicurezza che concerne tale sistema crittografico dipende unicamente dal mantenere la chiave privata nascosta mentre, al contrario, la chiave pubblica non compromette in alcun modo la sicurezza di una comunicazione.

1.3.3 Funzioni hash

La funzione hash è un algoritmo matematico che mappa i dati di lunghezza arbitraria (il messaggio) in una stringa binaria di dimensione fissa, chiamata *valore di hash* o *message digest*.
La resistenza alle collisioni delle funzioni hash si divide in due proprietà:

- Resistenza alle collisioni debole;
- Resistenza alle collisioni forte.

Queste due proprietà corrispondono a due diversi tipi di attacchi:

- Attacco a forza bruta: è un tipo di attacco basato sulla creazione di due diversi messaggi (M e M'), i quali producendo lo stesso valore hash fanno sì che l'attaccante, una volta che il mittente avrà firmato il messaggio M, ottenga la firma del messaggio M';
- Attacco del compleanno: questo attacco è basato sul paradosso del compleanno. Il mittente si prepara a *firmare* un messaggio, aggiungendo il codice hash di m bit appropriato e crittografando tale codice con la propria chiave privata. Successivamente, l'attaccante genera due elevato m diviso due varianti del messaggio, ognuna delle quali ha fondamentalmente lo stesso significato e, allo stesso tempo, genera un ugual numero di messaggi che sono varianti del messaggio fraudolento da sostituire a quello originale. I due insiemi di messaggi generati vengono confrontati fino a che non si trovano due messaggi che producano lo stesso valore hash. Trovata la coppia, l'attaccante offre la variante al mittente che, accettando la modifica e firmando il messaggio, starà firmando anche il messaggio fraudolento, essendo i messaggi con lo stesso codice hash in grado di produrre la stessa firma.

Caratteristiche di una funzione hash sicura

Le carattiristiche fondamentali che rendono sicura una funzione hash sono:

1. Può essere applicata a messaggi M di qualsiasi lunghezza producendo un output di lunghezza fissa h;

2. Deve essere facile da calcolare $h = H(M)$ per qualsiasi messaggio M;

3. Deve essere deterministica, cioè da input uguali si ottengono sempre output uguali (coerenza);

4. Avendo h, è computazionalmente impossibile determinare una M tale che $H(M) = h$ (one-way, cioè non invertibile);

5. Per qualsiasi blocco x è computazionalmente impossibile trovare una y diversa da x tale che $H(y) = H(x)$ (resistenza debole alle collisioni);

6. È computazionalmente impossibile trovare una coppia (x, y) tale che $H(x) = H(y)$, (univocità, resistenza forte alle collisioni);

7. Output uniformemente distribuito (casualità).

Algoritmi di hash

La struttura utilizzata dalla maggior parte delle funzioni hash è quella ideata da Merkle, che consiste in una funzione hash iterata. La funzione hash prende un messaggio in input e lo divide in L blocchi di dimensioni fisse pari a b bit.

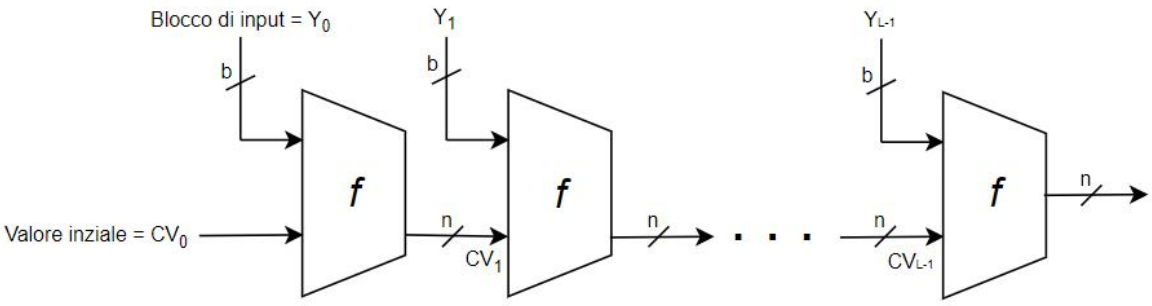

Figura 1.3: Funzione iterata di Merkle

L'algoritmo di hash comporta la ripetizione di una funzione di compressione f, che accetta due input (la variabile di concatenamento CV e un blocco Y di b bit) e produce

un output di n bit. All'inizio del calcolo della funzione hash, la variabile di concatenamento ha un valore prefissato dell'algoritmo (valore iniziale CV_0). Ad ogni iterazione CV_i corrisponderà una funzione $f(CV_{i-1}, Y_{i-1})$. Se la funzione di compressione è resistente alle collisioni precedentemente viste lo sarà anche la funzione iterata.
Una delle funzioni hash utilizzate di frequente nelle tecnologie blockchain è SHA (Secure Hash Algorithm). Tale algoritmo crittografico è stato sviluppato dall'NSA (National Security Agency) e dal NIST (National Istitute of Standards and Technology) e ha l'obiettivo di generare hash e codici univoci basati su uno standard con il quale documenti o dati informatici potrebbero essere protetti da qualsiasi agente esterno che desideri modificarli.

1.4 Blockchain

Una blockchain è un database distribuito a prova di manomissione che mantiene i record transazionali (dati), i quali sono raggruppati in blocchi. Un blocco è collegato al precedente includendo un identificatore univoco basato sui dati del blocco precedente. In funzione di ciò, se i dati vengono modificati in un blocco cambia l'identificatore univoco, che può essere visto in ogni blocco successivo. Questo effetto domino consente a tutti gli utenti all'interno della blockchain di sapere se i dati di un blocco sono stati manomessi. Poiché una rete blockchain è difficile da alterare o distruggere, essa garantisce un alto grado di resilienza.

1.4.1 Elementi chiave di una Blockchain

DLT (Distributed Ledger Technology)

DLT è un sistema di archiviazione delle transazioni che vengono archiviate in varie posizioni contemporaneamente. A differenza dei database tradizionali, la DLT non ha alcuna funzionalità centralizzata. Grazie alla crittografia, la DLT consente

un'archiviazione sicura e, una volta che le informazioni vengono salvate, diventino *immutabile*.

Record immutabili

Una volta che una transazione è immagazzinata nel registro, nessun utente può modificarla. Se la transazione deve essere modificata, deve essere creata una nuova transazione. In pratica, non si possono manipolare i dati presenti all'interno della blockchain.

Smart Contract

Uno smart contract è un protocollo automatico che viene eseguito e applicato. Esso memorizza e applica le clausole contrattuali tramite blockchain ed è regolato dai termini e condizioni espliciti. Negli Smart contract, quando un utente rispetti le condizioni specificate, il codice viene eseguito da solo ed è verificato mediante il meccanismo del consenso. I protocolli Smart contract sono decentralizzati, autonomi e trasparenti.

Senza fiducia (Trustless)

Essendo la blockchain un sistema decentralizzato, non è necessario fare affidamento su una terza parte per la sicurezza dei propri fondi. La blockchain è protetta da una rete decentralizzata peer-to-peer di *miners* (*minatori*) nota come Proof of Work, che elimina la necessità di fidarsi di qualsiasi intermediario.

1.4.2 Panoramica delle componenti

Si vogliono definire tutte le componenti principali di una blockchain:

- Nodo: è un dispositivo connesso a una rete peer-to-peer. Diffonde le transazioni e i blocchi in tutti gli altri nodi della rete;
- Transazione: è l'operazione che viene effettuata sul database;

- Blocco: all'interno di ciascun blocco vengono inserite tutte le transazioni recenti. Una volta che il blocco è terminato, viene aggiunto alla blockchain permanentemente;
- Catena: i blocchi vengono concatenati in modo ordinato, l'insieme dei quali forma una catena;
- Miners: sono nodi che verificano i blocchi prima di consentirne l'aggiunta alla struttura della blockchain;
- Consenso: Garantisce l'accordo tra ciascun nuovo blocco aggiunto e tutti i nodi della blockchain. Di conseguenza, un algoritmo di consenso cerca di trovare un accordo condiviso a vantaggio dell'intera rete.

Più avanti, verranno illustrate nello specifico queste ed altre componenti.

1.4.3 Tipi di Blockchain

Blockchain pubblica

Una blockchain pubblica (o permissionless) è la tipologia che può essere utilizzata per costruire una blockchain completamente aperta, come Bitcoin, e in grado di consentire a qualunque utente di unirsi e contribuire alla rete. Le operazioni in corso su una rete blockchain pubblica possono essere lette, scritte e verificate da chiunque, il che aiuta a preservarne l'integrità. Questo tipo di blockchain fornisce diversi vantaggi e svantaggi. Essi sono:

- Vantaggi:
  - È aperta a tutti;
  - Offre una maggiore garanzia di fiducia a tutti gli utenti;
  - Definisce un'alta sicurezza;
- Svantaggi:

  - Esecuzione lenta delle transazioni;
  - Difficile da scalare;
  - Dal punto di visto energetico non è efficiente;

Blockchain privata

Una blockchain permissioned è una blockchain privata in cui vengono applicate alcune restrizioni sugli utenti che possono accedere alla rete. Quest'ultima è controllata da una o più persone, il che richiede la dipendenza da terze parti per l'esecuzione delle transazioni. Solo le parti coinvolte sono messe a conoscenza delle transazioni in una blockchain privata. Seguono alcuni tra gli aspetti a favore e contro di questa seconda tipologia:

- Vantaggi:
  - Risultati in tempi estremamente veloci;
  - La rete è facilmente scalabile;
  - Energicamente è efficiente;
- Svantaggi:
  - Non è propriamente decentralizzata;
  - Poco trasparente;
  - Parzialmente immutabile;

Consorzio

Una blockchain federata è conosciuta anche come consorzio. Si tratta di una tecnologia blockchain in cui diverse entità controllano la rete. Il consorzio è un modello ibrido di blockchain pubblica e blockchain privata.

1.4.4 Funzionamento

Si vuole illustrare in definitiva come funziona sostanzialmente una blockchain. Per spiegarne al meglio il funzionamento, si prenda ad esempio una transazione nella quale viene effettuato una scambio di bitcoin tra due utenti. Poiché Bitcoin utilizza la prova crittografica al posto dell'attuale meccanismo "*trust-in-the-third-party*", viene impiegata la crittografia asimmetrica per firmare digitalmente una transazione. Una volta che la transazione viene generata, sarà visibile a tutti i nodi della rete e, successivamente, essa dovrà essere validata, come detto in precedenza, da una determinata quantità di nodi della rete. Infine, una volta che la transazione è stata validata, essa viene inserita nel registro condiviso, la blockchain appunto. Prima di registrare qualsiasi transazione nel registro, si devono garantire due caratteristiche:

1. L'utente che deve inviare i bitcoin deve effettivamente possedere tale criptovaluta;

2. Il mittente deve avere una sufficiente quantità di bitcoin nel suo account. Tale controllo viene effettuato tramite la verifica di tutte le precedenti transazioni dell'utente. Questo procedimento è di fatti volto a garantire che, prima di finalizzare una transazione, l'account contenga sufficiente criptovaluta per completare l'operazione.

Nella figura 1.4 viene definita la gestione di una transazione tra due utenti, Alice e Bob. Alice genera la transazione e, in seguito, la transazione viene rappresentata come un blocco. Quest'ultimo, nel momento in cui viene validato, è aggiunto alla catena dei blocchi precedentemente validati.

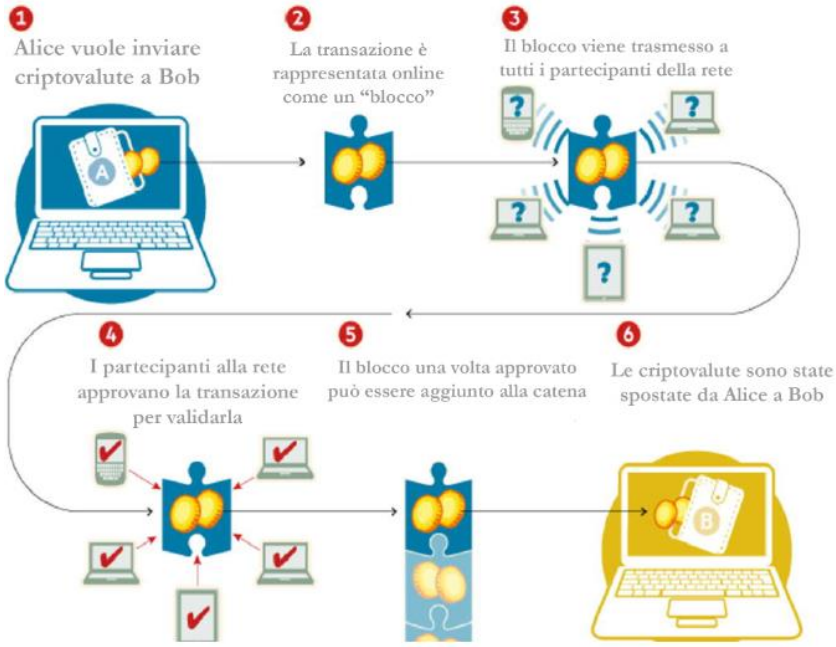

Figura 1.4: Esempio di una transazione tra due utenti.

1.4.5 Blockchain esistenti

Nell'esempio precedente è stato descritto il ciclo di vita di una transazione nella blockchain di bitcoin. Esistono tuttavia diversi tipi di blockchain, come Ethereum o Ripple.

Circa il primo, il sito ufficiale definisce Ethereum una piattaforma decentralizzata che gestisce gli "*Smart Contract*". La principale differenza tra Bitcoin e Ethereum è che, quest'ultima permette di generare mediante codice la criptovaluta (ether) risultandone, quindi, la creazione più facile e veloce. In merito al secondo, è possibile definire la blockchain Ripple come "*ibrida*". Essa rende possibile l'invio di denaro sfruttando le capacità fornite dall'architettura basata sul modello Shared Decentralized Ledger. Le istituzioni finanziare che entrano a far parte della rete Ripple possono processare pagamenti ai propri clienti, in qualsiasi parte del mondo, in modo istantaneo, affidabile ed economico.

1.4.6 Componenti blockchain

Sebbene ad un primo approccio possa sembrare complessa, prendere in considerazione ed esaminare le componenti di una blockchain singolarmente può risultare utile ai fini di una migliore comprensione.

Funzione hash

La funzione hash che viene spesso utilizzata nella maggior parte delle blockchain è la SHA256. All'interno di una rete blockchain, le funzioni hash vengono impiegate per svolgere diverse attività, come:

- Derivazione dell'indirizzo;
- Creazione di identificatori univoci;
- Sicurezza del blocco dati: una volta stabilito un nodo, sarà eseguito l'hash del blocco dati, generando a suo volta un hash che verrà immagazzinato all'interno del blocco header;
- Sicurezza del blocco header: una volta stabilito un nodo, sarà eseguito l'hash del blocco header. Se, ad esempio, la rete utilizza il modello di consenso *proof-of-work*, l'hash del blocco header verrà semplicemente generando tramite un differente valore *nonce*. Una volta creato l'hash del blocco header, l'intestazione del blocco corrente verrà inclusa all'interno del blocco successivo, dove saranno protetti i dati del blocco header corrente.

Transazioni

Una transazione rappresenta, come si è visto, un'interazione tra utenti della rete. Ogni blocco della blockchain può contenere zero o più transazioni e i dati contenuti in quest'ultime possono essere differenti per ogni blockchain; per quanto il meccanismo delle transazioni è in gran parte lo stesso. Un utente delle rete blockchain invia informazioni alla rete stessa. L'informazione inviata, tipicamente, include l'indirizzo

del mittente, la sua chiave pubblica, la firma digitale, gli input e gli output della transazione. Una singola transazione di criptovaluta generalmente richiede le seguenti informazioni (ma può contenerne di più):

- Input - si tratta di una lista di risorse (asset) digitali da trasferire. Una transazione prevede il riferimento alla fonte, cioè alla provenienza della risorsa digitale, o, in altri termini, un riferimento agli eventi passati dell'asset. Il mittente deve fornire la prova che può avere accesso agli input di riferimento, e quasi sempre ciò avviene firmando digitalmente la transazione;
- Output - sono generalmente gli account che fanno da recipiente per gli asset digitali forniti o ricevuti. Ogni output specifica il numero di asset digitale trasferiti al nuovo proprietario, l'identificatore del nuovo proprietario e un set di condizioni che i nuovi proprietari dovranno soddisfare per poter utilizzare quelle risorse.

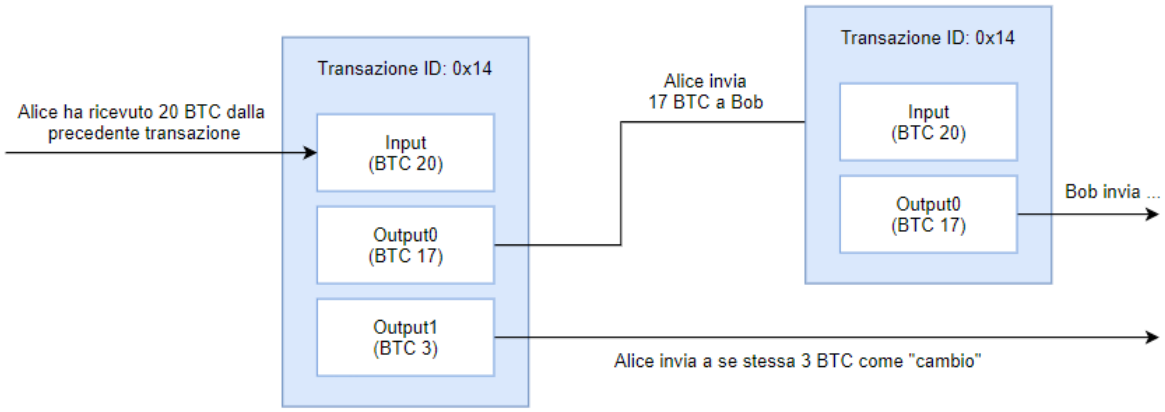

Figura 1.5: Esempio di transazione con bitcoin

Una volta effettuata una transazione, è importante determinarne la validità e l'autenticità. La validità indica che la transazione soddisfa i requisiti del protocollo adottato dalla blockchain. L'autenticazione è altresì importante perché determina che il mittente dell'asset digitale ha veramente accesso agli asset inviati.

Derivazione degli address

La maggior parte delle blockchain fanno uso di un *address*, che è una stringa di caratteri alfanumerici. Questa stringa viene derivata mediante le funzioni hash. Un metodo per generare un address è quello di creare una chiave pubblica e su cui si applica un algoritmo hash. Il risultato sarà l'address.

Figura 1.6: Derivazione dell'address

Blocchi

Ogni volta che un nodo pubblica un blocco, a esso verranno aggiunte le transazioni. Un blocco è formato da un *blocco header* e un *blocco dati*. Molte blockchain utilizzano diversi campi all'interno di tali blocchi. Questi campi sono i seguenti:

- Blocco header
  - Numero di blocco, noto anche come altezza del blocco;
  - Valore hash del precedente blocco header;
  - Rappresentazione del blocco dati;
  - Timestamp;
  - Misura del blocco;
  - Valore nonce;
- Blocco dati
  - Lista delle transazioni e dei registri inclusi nel blocco;
  - Altri dati aggiuntivi opzionali.

Concatenamento dei blocchi

I blocchi sono concatenati insieme nel punto in cui ogni blocco contiene l'hash del precedente blocco header. L'insieme di questi blocchi forma la cosiddetta *block-chain* (letteralmente "catena di blocchi"). Nella figura 1.5 è rappresentata una generica catena di blocchi. È importante notare come i blocchi sono concatenati seguendo una linea temporale: con ciò il sistema evita di creare nuovi blocchi prima di aver raggiunto la misura massima dell'ultimo blocco.

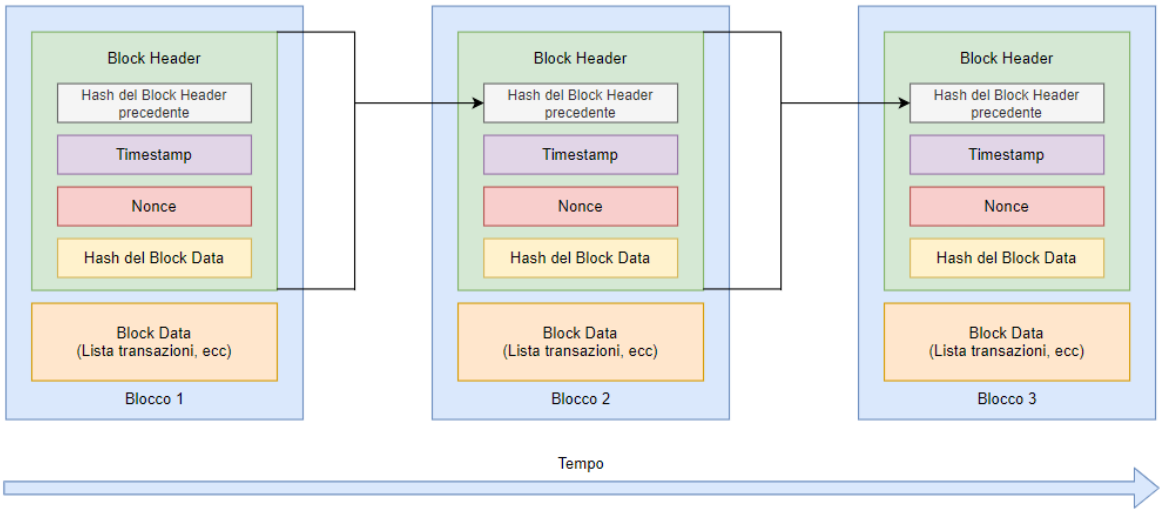

Figura 1.7: Blocchi e concatenamento dei blocchi

1.4.7 Algoritmi di consenso

Il consenso è un processo che consente a tutti i nodi di un sistema distribuito di concordare una decisione, grazie a un'insieme di protocolli e regole. Il consenso risulta molto complesso quando sono inclusi vari vincoli. Uno dei vincoli più complicati da gestire è il *Byzantine Fault*. Il byzantine fault dice che potrebbero esserci nodi anormali o dannosi che non seguono le regole o che addirittura le violano intenzionalmente per impedire il consenso. Teoricamente, è stato dimostrato che non esiste un algoritmo tollerante al byzantine fault ma all'aumentare del numero di nodi

di pari passo diminuisce la probabilità che esso si verifichi questo perché dovrebbero esserci un numero considerevole di nodi corrotti o guasti.

3m + 1 Processor algorithm

Il primo algoritmo che garantisce la tolleranza al Byzantine Fault è il *"3m + 1 Processor algorithm"* creato nel 1980. Questo algoritmo dimostra che in una rete sincrona con *m* nodi difettosi, non esiste alcun algoritmo di consenso tollerante al byzantine fault quindi per essere tollerante la rete deve avere almeno 3m + 1 nodi, dove m è il nodo guasto. Ad esempio se la rete ha tre nodi di cui uno difettoso la formula è $3 * 1 + 1$ mentre se i nodi guasti sono due contemporaneamente la rete per essere deve avere almeno sette nodi $(3 * 2 + 1)$.

Proof of Work

Nel precedente algoritmo i nodi collaborano per evitare il Byzantine fault però esso prevede un vincolo sul numero di nodi e richiedono periodiche comunicazioni, il che limita la scalabilità.
L'algoritmo Proof of Work (PoW) adotta un approccio completamente diverso. In una rete PoW, ogni nodo *"compete"* invece di collaborare. Il nodo che riesce a risolvere un quesito matematico estremamente difficile da risolvere ma molto facile da verificare prenderà una decisione per l'intera rete in quel momento. Subito dopo che un quesito è stato risolto e la soluzione si è propagata in tutta la rete, inizia la competizione per il quesito successivo. La risoluzione di un quesito è completamente probabilistica e dipende esclusivamente dalla potenza di calcolo. Quindi se la potenza è sufficientemente decentralizzata tra i nodi, è in realtà impossibile per uno o più nodi regnare su una rete PoW. Per le reti blockchain più famose come Bitcoin ed Ethereum implementano l'algoritmo Proof of Work. Il problema principale per queste reti è trovare un valore *nonce* che soddisfi alcune condizioni. L'azione di trovare un nonce che soddisfa e risolve un quesito è chiamata *mining*. Il nodo che riesce a trovare il nonce riceve una ricompensa quindi risolvere un quesito è come estrarre l'oro.

In Bitcoin ed Ethereum, ogni transazioni è firmata digitalmente in modo che un nodo guasto non possa danneggiare le transazioni. In questo modo, utilizzando sia il PoW che la firma digitale, Bitcoin ed Ethereum possono raggiungere la tolleranza al Byzantine fault. Poiché il PoW funziona in base alla concorrenza e non alla collettività, la complessità è irrilevante per il numero di nodi ma, naturalmente, più grande è il numero di nodi della rete e più essa risulta sicura.
L'algoritmo Proof of Work ha anche dei problemi come l'enorme spreco di energia dovuto alla forte concorrenza oltre problema ecologico che ne deriva oppure il problema della finalità (situazioni in cui viene eliminato un blocco per evitare la biforcazione della catena di blocchi).

1.4.8 Sicurezza nelle blockchain

L'utilizzo della tecnologia blockchain non rimuove i rischi inerenti alla sicurezza informatica, i quali richiedono una gestione del rischio ponderata e pro attiva. Un solido programma di sicurezza informatica rimane vitale per proteggere la rete e le organizzazioni partecipanti dalle minacce informatiche.
Fatta questa premessa, gli standard e le linee guida esistenti per la sicurezza si confermano altamente affidabili per garantire la protezione dei sistemi che si interfacciano o che si basano su reti blockchain. In generale, gli standard e le linee guida esistenti permettono di avere una base solida per proteggere le reti blockchain dagli attacchi informatici.

Attacchi informatici basati sulla rete

Le tecnologie blockchain sono ben pubblicizzate come estremamente sicure, grazie alla progettazione a prova di manomissione: una volta che una transazione è immagazzinata in un blocco pubblico, generalmente non può più essere modificata. Tuttavia, questo è vero solo per le transazioni che sono state immagazzinate nel blocco pubblico. Le transazioni che non siano state ancora incluse nel blocco sono

vulnerabili a diversi tipi di attacchi. Per le reti blockchain che, ad esempio, si servono di timestamp transazionali, questo potrebbe avere risvolti positivi ma anche, e sopratutto, negativi su una transazione. Gli attacchi DoS (Denial of Service) possono essere condotti sulla piattaforma blockchain o negli smart contract implementati sulla piattaforma.
Infine, le reti blockchain e le loro applicazioni non sono immuni da utenti malintenzionati che possono condurre scansioni della rete per scoprirne e sfruttarne le vulnerabilità.

Utenti malintenzionati

La preoccupazione maggiore degli utenti malintenzionati è ottenere abbastanza potenza per causare danni alla rete. Se malauguratamente questa potenza venisse raggiunta, l'utente potrebbe causare danni molto ampi alla rete, tra cui:

- Oscurare le transazioni di utenti, nodi o persino di interi paesi specifici;
- Creare una catena alternativa in segreto. Una volta che la catena alternativa è più lunga della catena reale, i nodi onesti passeranno alla catena con maggiore lavoro svolto (comportamento che è peculiare della tecnologia blockchain). Questo tipo di attacco compromette il principio per cui una rete blockchain è a prova di manomissione.

CAPITOLO 2

Obiettivi e soluzioni

Il presente lavoro mira allo sviluppo di un sistema di certificazione digitale che sfrutti le potenzialità della tecnologia blockchain. In particolar modo, la scelta per il tipo di blockchain ricade sul modello permissioned, poiché il sistema deve permettere l'accesso esclusivamente a partecipanti verificati e conosciuti. Il sistema dovrà inoltre consentire agli stessi di caricare documenti che certificano l'assegnazione o l'acquisizione di titoli (ad esempio attestati, diplomi, ecc.). Data la scelta di una soluzione blockchain permissioned, i documenti caricati su di essa sono verificati da tutti i partecipanti alla rete.

Le principali entità del sistema sono: l'Università (proprietaria del sistema) e gli enti esterni (aziende, enti certificativi, ecc.). Ogni ente, prima di poter partecipare alla rete, deve ricevere l'autorizzazione dall'università. L'offerta erogata agli enti prevede una serie di funzionalità tra cui quella di creare eventi (corsi, congressi, ecc.) interni all'università. A conclusione di ogni evento, l'ente potrà rilasciare un attestato (documento) a ciascun utente che vi avrà partecipato. Ogni operazione permessa ai partecipanti dovrà essere specificata nella regole della rete, cioè nello

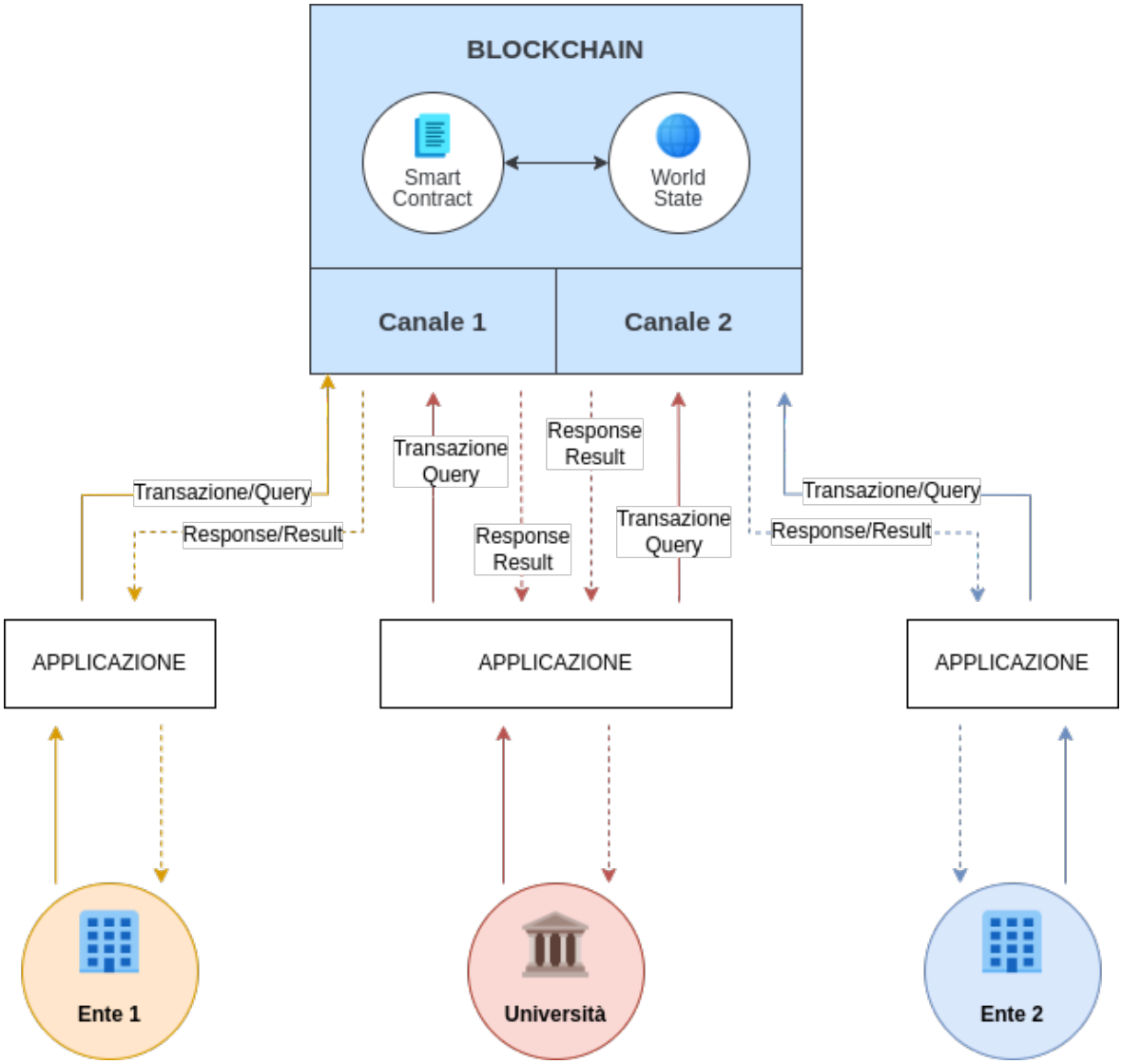

Figura 2.1: Architettura semplificata.

smart contract. Un altro vantaggio derivante dall'impiego di una blockchain è la proprietà di mantenere un sistema "protetto" da verifiche crittografiche, pertanto il più adeguato per la gestione di dati che devono rimanere sotto lo stretto controllo dell'università.

In figura 2.1 si rappresenta un'architettura semplificata del sistema che verrà implementato. Nello specifico, vengono definite tre entità principali nel sistema: Università, Ente 1 ed Ente 2. Ogni entità del sistema potrà cambiare il *world state* (ossia modificare lo stato della blockchain con scritture e letture sul registro) attraverso operazioni definite all'interno degli smart contract. In secondo luogo, vengono specificati due

canali nei quali le entità potranno comunicare: canale 1 e canale 2. Nel primo canale potrà comunicare il primo Ente con l'Università, nel secondo canale avverrà la comunicazione tra il secondo Ente e l'Università. È importante notare che in ogni caso agli Enti non è permesso di comunicare.
Ai fini raggiungere un certo livello di completezza, verranno approfondite e applicate diverse tecnologie, tra cui:

- Hyperledger Fabric: una piattaforma DLT (Distributed Ledger Technology) open-source che offre alcune chiavi di differenziazione rispetto ad altri progetti blockchain;
- InterPlanetary File System (IPFS): un sistema distribuito per l'archiviazione e l'accesso a file, siti Web, applicazioni e dati. Poiché all'interno della blockchain non è consentito l'inserimento di file, questi verranno caricati su IPFS, che produrrà un valore hash. Tale valore hash, al momento della scrittura, sarà inserito sulla blockchain all'interno della transazione;
- Flask: un micro-framework Python capace di creare un'API per comunicare con il database, Fabric e IPFS.

CAPITOLO 3

Progettazione

Questa fase renderà semplice, efficiente e ordinata la fase di implementazione. Inizialmente vengono definiti gli attori che interagiscono con il sistema; seguono il disegno e l'analisi dell'architettura adatta da implementare. Infine, in base alle specifiche definite dall'architettura, vengono progettate tutti le singole componenti.

3.1 Attori

Allo scopo di capire al meglio le funzionalità da sviluppare per un sistema di certificazione digitale, che gestisca documenti (titoli accademici, certificazioni, ecc.) rilevanti in ambito accademico, è fondamentale definire gli attori che vanno a interagire con il suddetto sistema. Principalmente sono tre gli attori presenti nel sistema:

- L'università: potrà effettuare operazioni di gestione e amministrazione sul sistema. L'operazione più importante svolta dall'università (admin) è la verifica e la gestione dei documenti che circolano all'interno di un istituto accademico;

- L'ente: ha la possibilità di pianificare eventi (corsi, workshop, congressi, ecc.) erogabili per gli utenti. Le operazioni principali dell'ente sono la gestione dei titoli (rilascio, revoca e aggiornamento) e la gestione delle attività (creazione, modifica ed eliminazione);

- L'utente: rappresenta tutti gli studenti, i docenti, il personale amministrativo e il personale tecnico. Le principali funzionalità degli utenti sono la gestione delle richieste suddivisa in: richiesta attribuzione titoli e richiesta di partecipazione.

Oltre alle precedenti operazioni definite, utenti, enti e università condividono la gestione dell'accesso al sistema, quindi log-in e log-out saranno condivisi da ogni attore. Nella figura 3.1 è stato formulato un diagramma dei casi d'uso che generalizza gli attori e le operazioni che essi possono eseguire sul sistema.

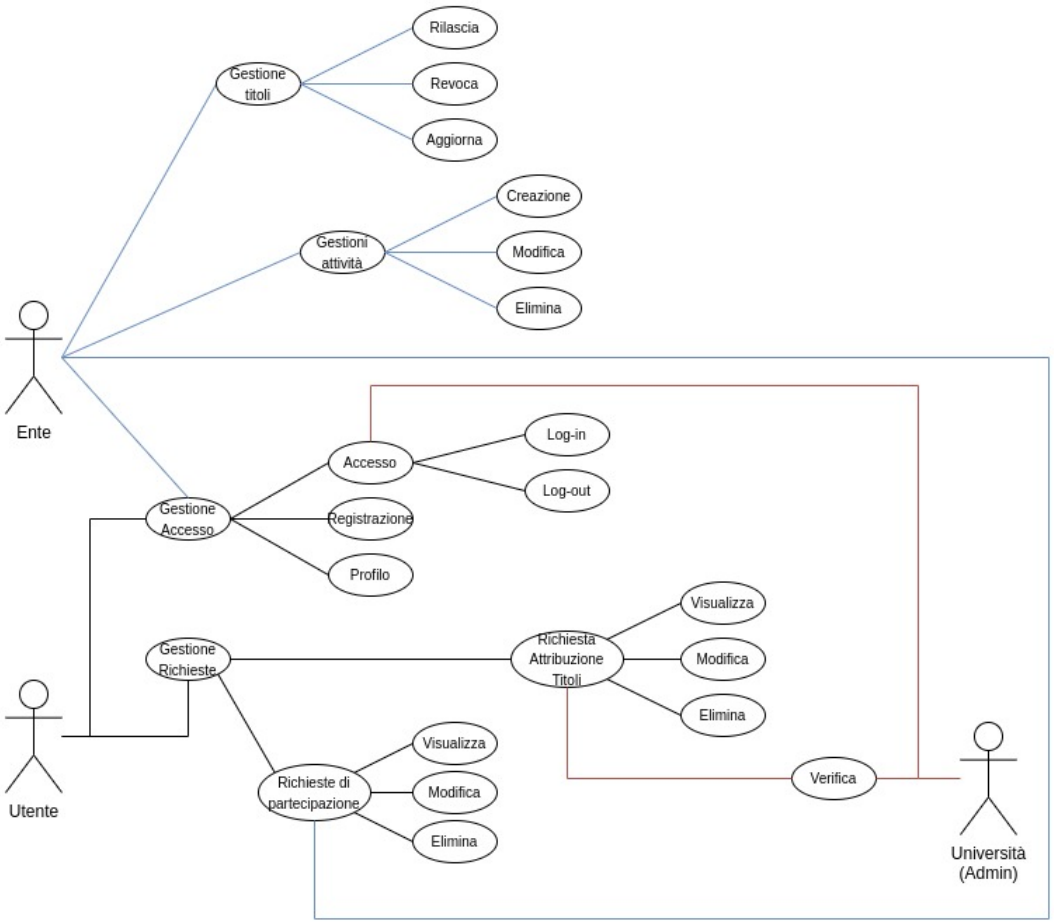

Figura 3.1: Use case diagram

3.2 Architettura

Il secondo passo è la progettazione dell'architettura del sistema. Per le sue fondamenta viene utilizzata un'architettura client-server. Essa è suddivisa in due parti: il client e il server.
Il client al suo interno contiene gli attori del sistema e l'interfaccia utente. Lo scopo principale di ogni interfaccia utente è quella di permettere agli utenti di interagire facilmente con il sistema. Ogni interazione dell'utente potrà generare una richiesta (es. HTTP request) al server.
Il server, per ogni richiesta ricevuta dal client, tramite un'API (Application Programming Interface) processa la richiesta e successivamente invia una risposta (es. HTTP response) al client. Tra le componenti del server vi sono:

- Un database: per contenere tutte i dati e le informazioni del sistema;
- Una blockchain: quando vengono caricati dei documenti nel sistema, deve essere possibile verificarli e renderli immutabili (appunto tra le principali funzioni attribuite alla blockchain);
- Un sistema di file storage: sia all'interno del database che all'interno della blockchain non vengono caricati i titoli (es. documenti in pdf). Nel sistema di file storage viene caricato il documento che genera un identificativo, il quale sarà inserito all'interno del database. Tramite l'identificativo è possibile risalire al documento ogni qualvolta ce ne sarà bisogno.

Nella figura 3.2 viene rappresentata una versione semplificata di quanto appena descritto.

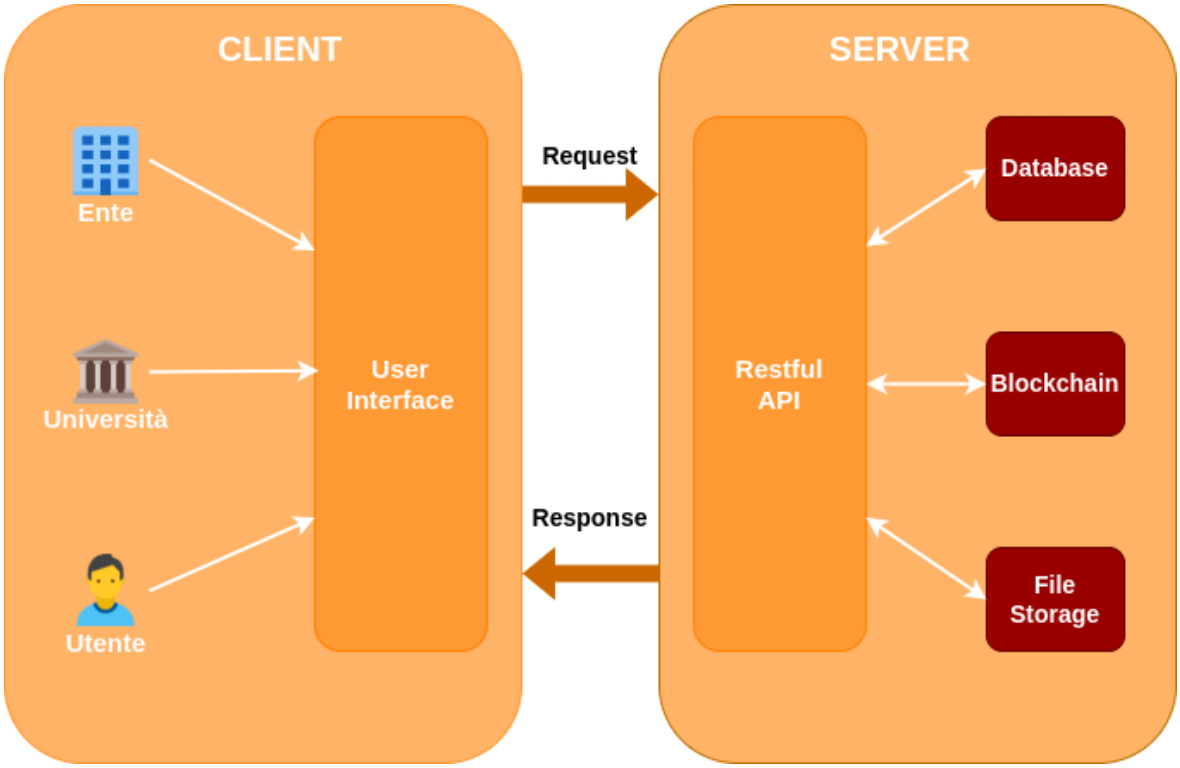

Figura 3.2: Rappresentazione dell'architettura client-server

3.3 Database

Per la progettazione del database viene utilizzato il tipo relazionale, in quanto risulta evidente che ogni dato deve essere in relazione con gli utenti del sistema. Questa sezione è suddivisa in due parti:

1. Progettazione concettuale: traduce le problematiche affrontate in uno schema concettuale facile da comprendere. Il modello ER (Entità-Relazione) è adottato durante questo tipo di progettazioni per definire i dati del sistema;

2. Progettazione logica: il cui obiettivo è quello di tradurre il modello ER (Entità-Relazione) in uno schema logico che mostri come effettivamente il database deve essere implementato, indipendentemente dal DBMS (DataBase Management System) adottato. Tale fase si compone di due ulteriori passaggi:

   (a) Ristrutturazione del modello Entità-Relazione: serve per ottimizzare il modello logico;

(b) Traduzione verso il modello logico: fa riferimento ad uno specifico modello logico, in questo caso quello relazionale.

3.3.1 Progettazione concettuale

È possibile costruire il diagramma ER (Entità-Relazione), in rapporto alle informazioni dei capitoli precedenti. Il diagramma ER è rappresentato nella figura 3.3.

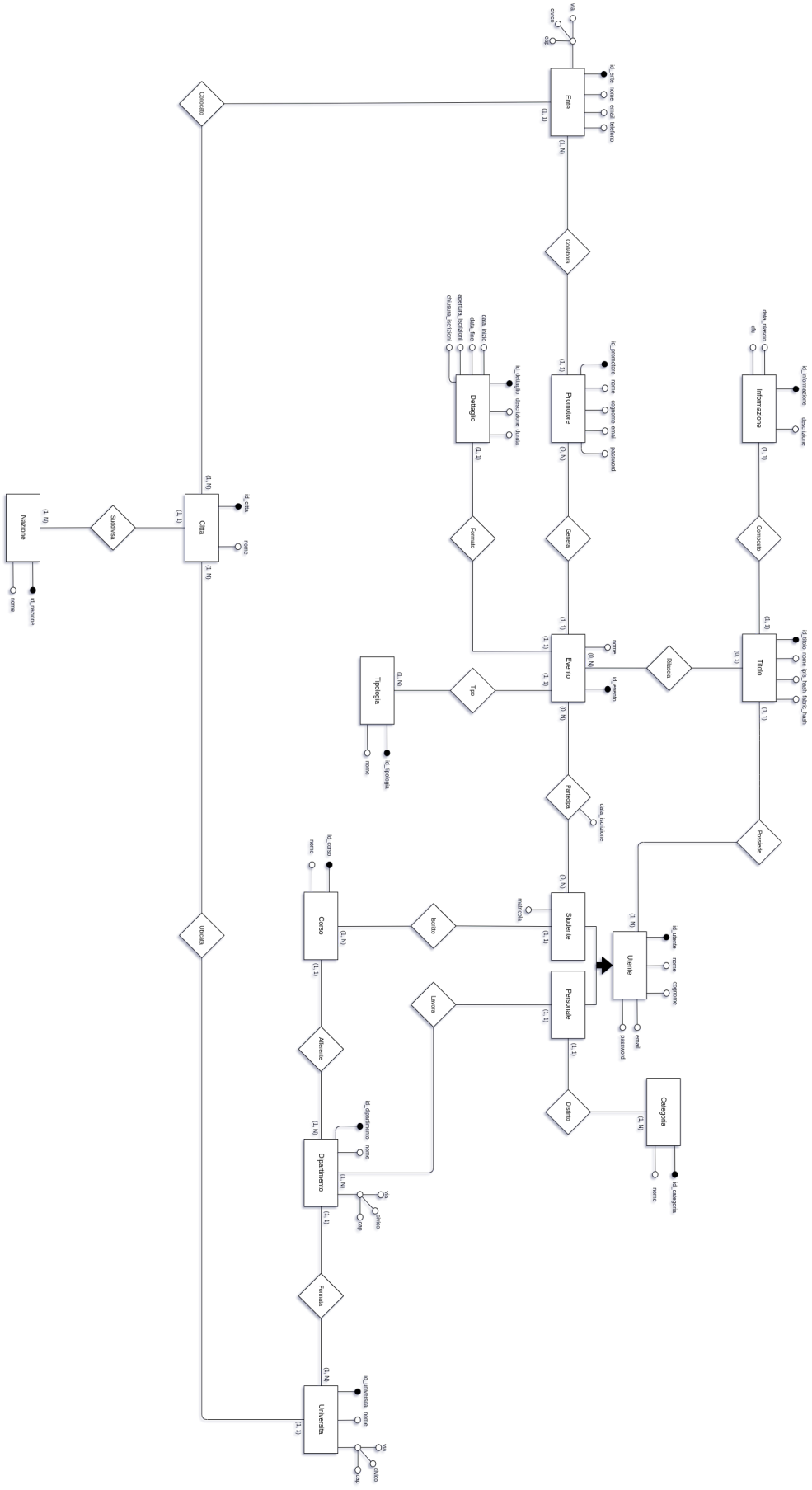

Figura 3.3: Diagramma Entità-Relazione

3.3.2 Progettazione logica

Ristrutturazione del modello Entità-Relazione

L'operazione di ristrutturazione del modello ER si realizza mediante:

- L'analisi delle ridondanze: tale operazione viene giudicata irrilevante per il raggiungimento delle finalità del presente lavoro, quindi esclusa dallo stesso;
- Eliminazione delle generalizzazioni;
- Identificazione delle chiavi primarie.

Analizzando il diagramma Entità-Relazione è possibile notare un'unica generalizzazione, quella dell'entità Utente a cui sono associate due entità figlie: Studente e Personale. Per rimuovere questa generalizzazione si è scelto di eliminare l'entità Utente e mantenere le entità figlie.
La fase di identificazione delle chiavi primarie non si è dovuta svolgere poiché non sono presenti chiavi composte in alcuna entità.
Il risultato della fase di ristrutturazione è rappresentato dalla figura 3.4.

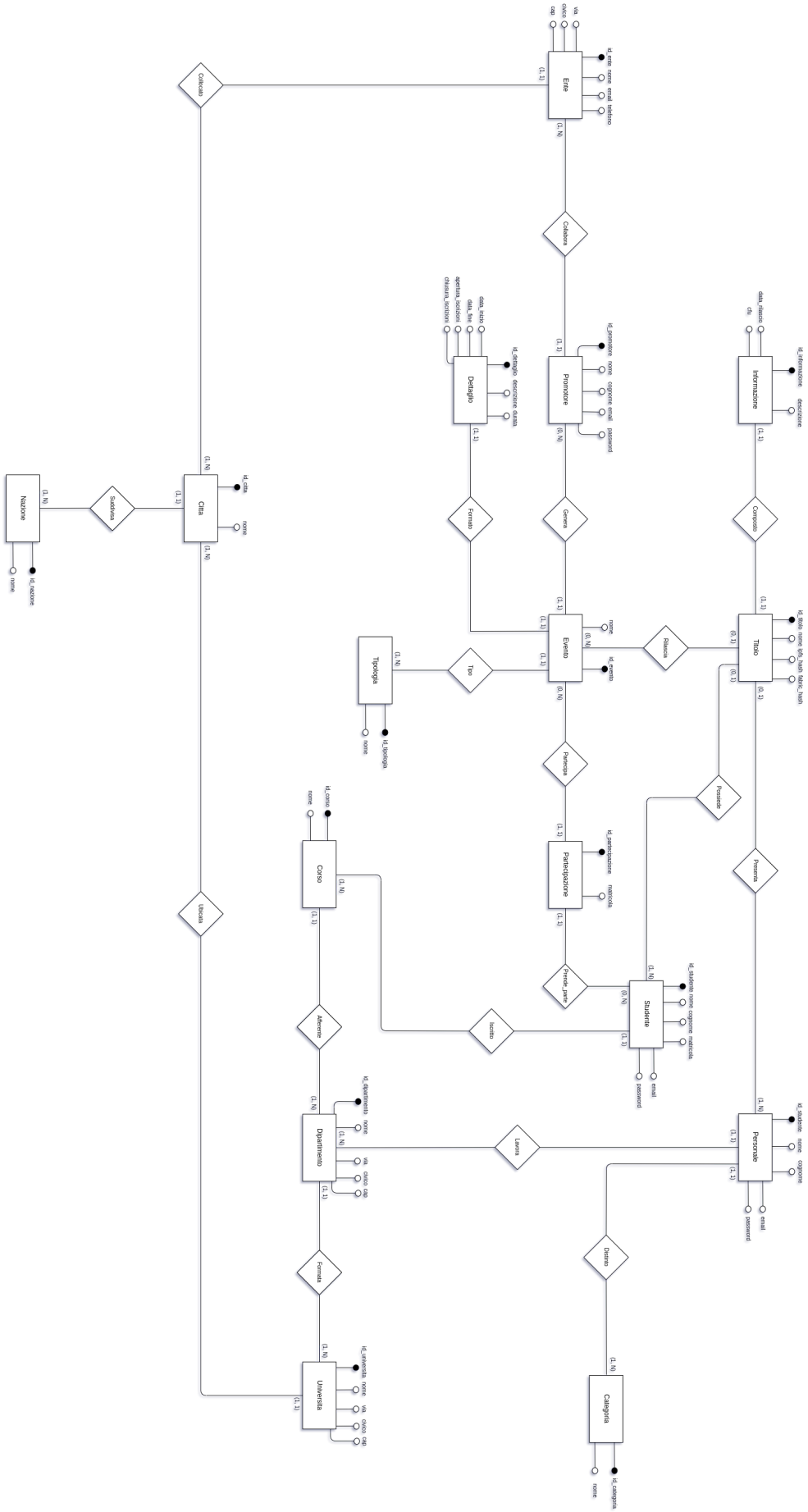

Figura 3.4: Diagramma Entità-Relazione ristrutturato

Traduzione in modello logico

Una volta completata la ristrutturazione del modello Entità-Relazione si passa alla traduzione del modello logico. Il modello logico è il modo in cui devono essere strutturate le tabelle, cioè: il nome della tabelle, le chiavi primarie, eventuali chiavi esterne e gli attributi rimanenti.

Tabella	Chiave primaria	Chiave esterna	Attributi
Studente	id_studente	id_corso	nome, cognome, email, password, matricola
Corso	id_corso	id_dipartimento	nome
Dipartimento	id_dipartimento	id_universita	nome, via, civico, cap
Universita	id_universita	id_citta	nome, via, civico, cap
Citta	id_citta	id_nazione	nome
Nazione	id_nazione		nome
Personale	id_personale	id_dipartimento, id_categoria	nome, cognome, email, password
Categoria	id_categoria		nome
Titolo	id_titolo	id_evento, id_personale, id_studente	nome, ipfs_hash, fabric_hash
Informazione	id_informazione		descrizione, data_rilascio
Evento	id_evento	id_tipologia, id_promotore	nome

Dettaglio	id_dettaglio		descrizione, durata, data_inizio, data_fine, apertura_iscrizioni, chiusura_iscrizioni
Ente	id_ente	id_citta	nome, email, telefono, via, civico, cap
Tipologia	id_tipologia		nome
Partecipazione	id_partecipazione	id_evento, id_studente	data_iscrizione
Promotore	id_promotore	id_ente	nome, cognome, email, password

Tabella 3.1: Modello logico

3.4 Struttura logica del software

A questo punto può risultare utile descrivere in modo più o meno dettagliato le funzionalità delle diverse parti del sistema, così da dare una visione completa dell'interazione dei vari elementi. A tal proposito, ci si serve di alcuni diagrammi secondo lo standard fornito da UML (Unified Modeling Language).
In particolare, vengono utilizzati i diagramma di stato per descrivere due operazioni: gestione dell'accesso e gestione dei titoli. Viene definito anche un diagramma delle attività che serva come input per l'algoritmo di inserimento dei titoli sulla piattaforma da parte dell'Utente.

3.4.1 Diagrammi di stato

I diagrammi di stato danno un'idea chiara del funzionamento logico che dovrà avere il programma finale.
Nella figura 3.5 viene riportato il diagramma di stato per la gestione dell'accesso. Nello specifico, la gestione dell'accesso è uguale per tutti gli utenti del sistema. In

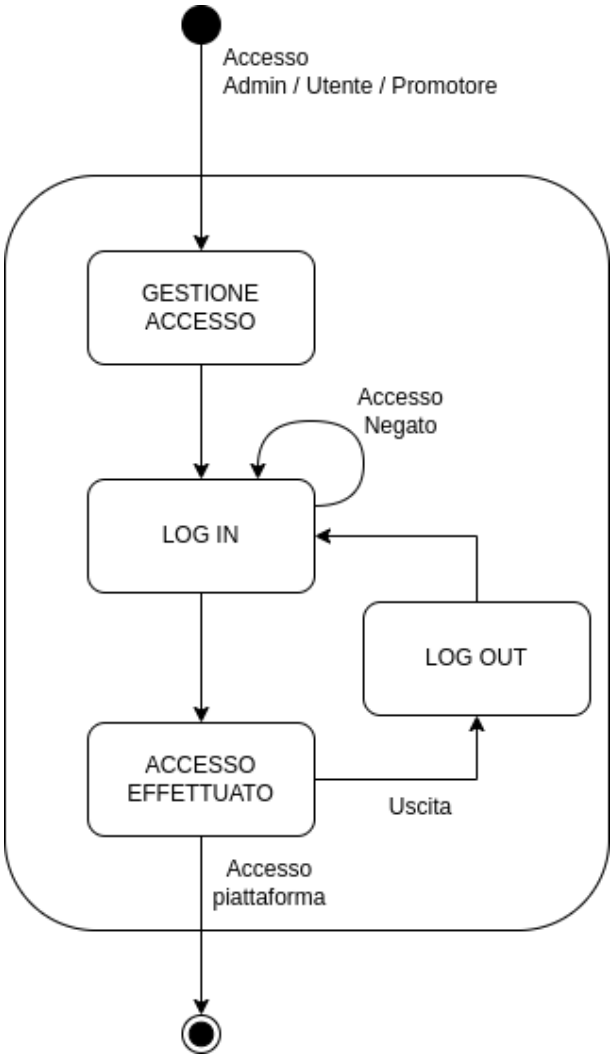

Figura 3.5: Diagramma di stato per la gestione dell'accesso.

seguito vengono effettuate le operazioni per la gestione dei *titoli*. In particolare, sono stati realizzati tre diagrammi di stato, che sono:

- Diagramma Admin (figura 3.6): l'Admin è colui che deve verificare i titoli inseriti dagli utenti e quelli rilasciati dagli enti. Una volta che un titolo viene verificato non potrà più essere modificato;

- Diagramma Utente (figura 3.7): rappresenta la possibilità per l'utente di inserire i propri titoli conseguiti, che non saranno autenticati fino a quando l'Admin non lo verificherà;

- Diagramma Promotore (figura 3.8): il promotore rappresenta l'ente di riferimento. Esso ha la possibilità di rilasciare dei certificati da parte dall'ente di cui fa parte.

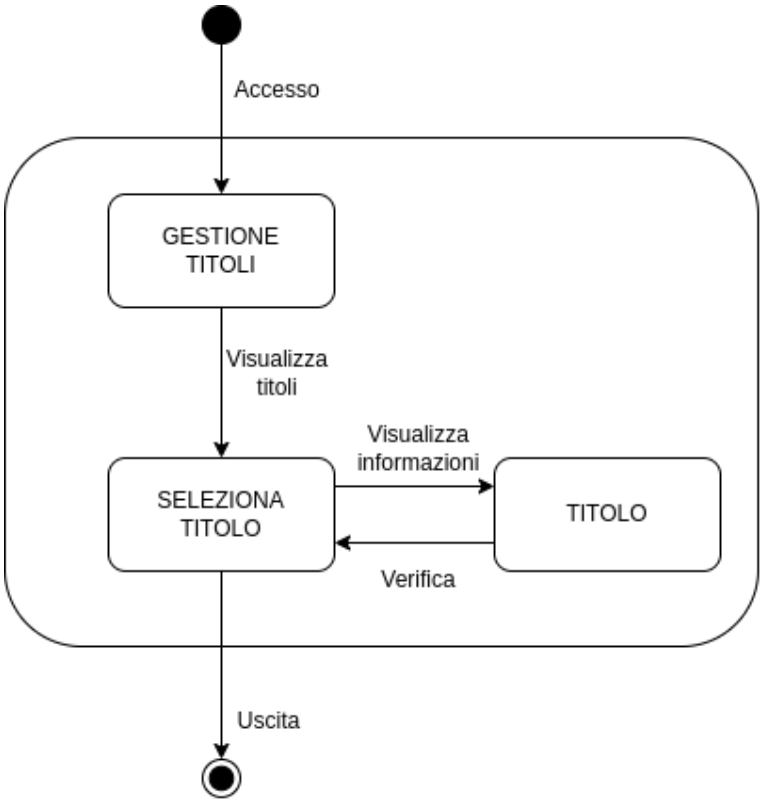

Figura 3.6: Diagramma di stato per la gestione dei titoli lato Admin.

Infine, nel diagramma delle attività (figura 3.9) viene rappresentato l'algoritmo di inserimento del titolo, come detto in precedenza. Quest'algoritmo prende in input alcuni parametri, ovvero:

- File: il documento in formato PDF (Portable Document Format) del titolo da aggiungere;

- Evento: è costituito dall'identificativo (un valore intero) dell'evento da cui è stato rilasciato il titolo (può essere un valore nullo);

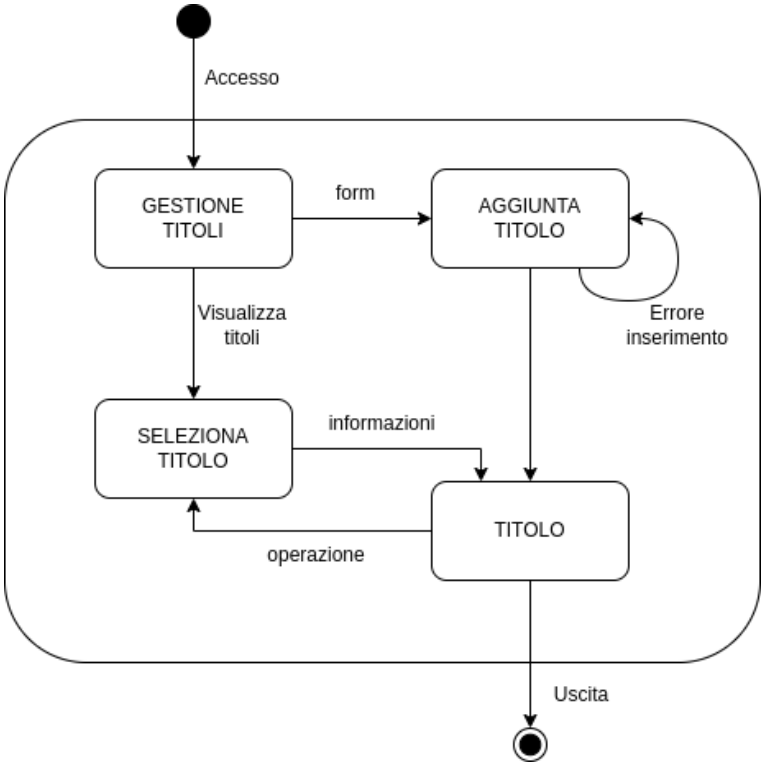

Figura 3.7: Diagramma di stato per la gestione dei titoli lato Utente.

- Utente: identificativo dell'utente a cui appartiene il titolo;
- Nome Titolo: indica il nome che rappresenta il titolo da inserire;
- Descrizione: un'insieme di informazioni aggiuntive per descrivere dettagliatamente il titolo;
- Data Rilascio: la data in cui l'Utente ha ottenuto il titolo.

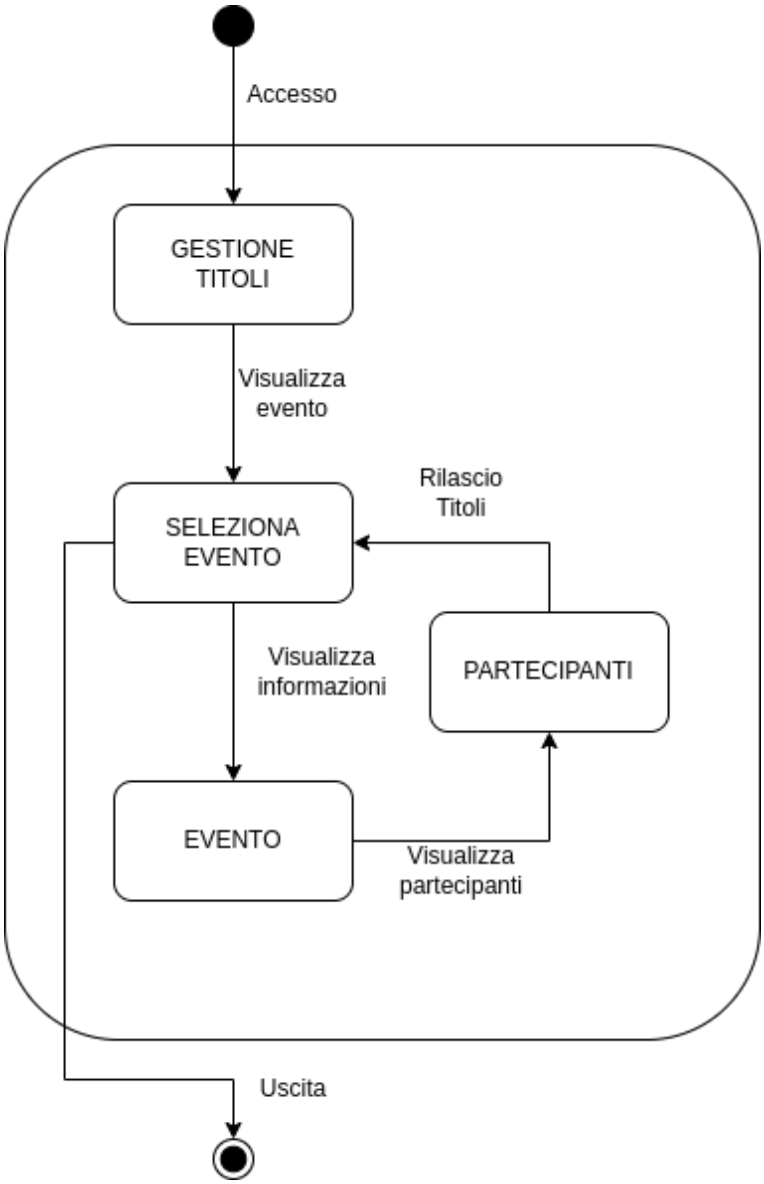

Figura 3.8: Diagramma di stato per la gestione dei titoli lato Promotore.

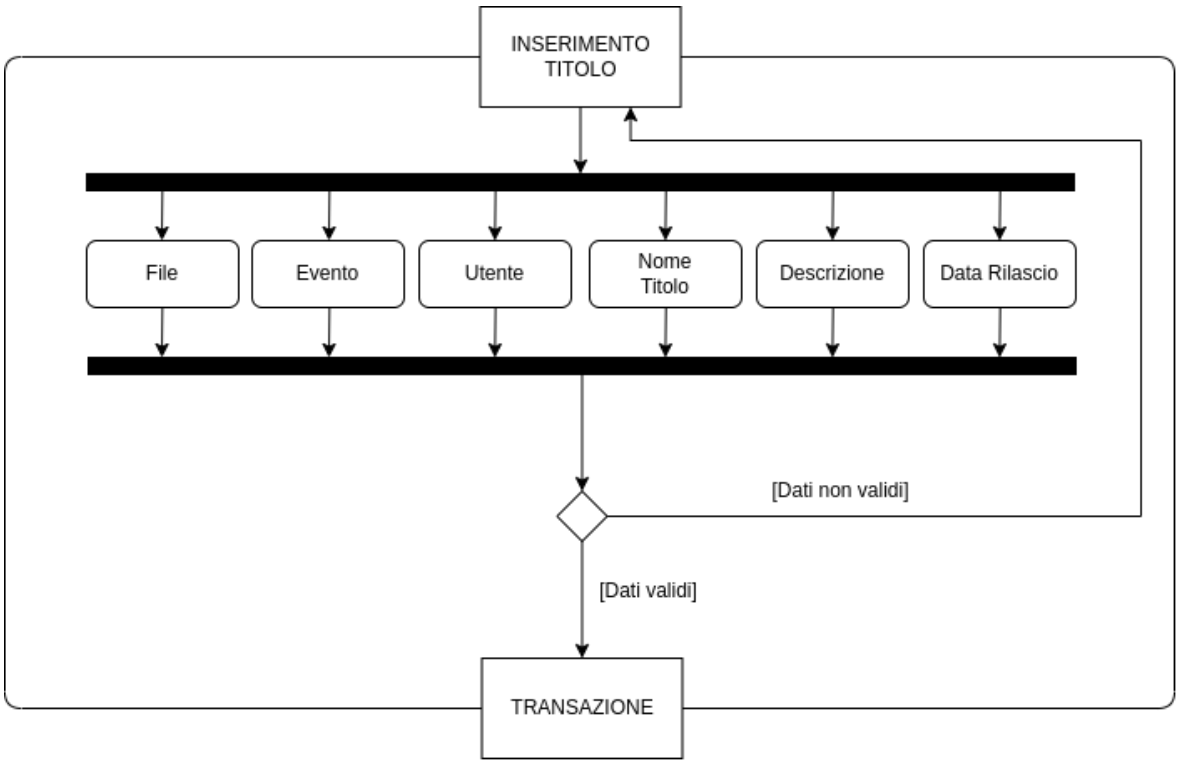

Figura 3.9: Diagramma delle attività dell'inserimento di un titolo.

CAPITOLO 4

Tecnologie abilitanti

In questo capitolo vengono esplicitate le principali tecnologie, i linguaggi di programmazione o framework impiegati per la realizzazione del presente studio.

4.1 Python

Python è un linguaggio di programmazione potente e facile da imparare. L'elegante sintassi e la tipizzazione dinamica di Python, insieme alla sua natura esplicativa, lo rendono il linguaggio ideale per lo scripting e lo sviluppo rapido di applicazioni in molte aree sulla maggior parte delle piattaforme esistenti.
La versione di Python che si è scelto di adottare per questo studio è python 3.8.10.
Tra i vantaggi associati a Python, troviamo:

- La popolarità, in quanto facile da codificare e capire;
- Seppur implementato come linguaggio di programmazione orientato agli oggetti, può essere utilizzato anche per scrivere codici funzionali;

Figura 4.1: Logo Python.

- L'introduzione sul mercato di un programma Python richiede molto meno tempo rispetto a programmi implementati in altri linguaggi;
- Un gran numero di comunità e libri sono disponibili per supportare gli sviluppatori Python;
- Non è necessario dichiarare i tipi di variabili, pertanto è più veloce implementare un'applicazione Python.

Naturalmente, presenta anche alcuni svantaggi:

- Python è più lento di altri linguaggi. Ciò è dovuto alla mancanza di ottimizzatori *Just In Time* in Python;
- Il vincolo dell'indentazione obbligatoria lo fa apparire leggermente spiacevole per i nuovi programmatori;
- Python non è adatto per sistemi di basso livello o di interazione hardware.

La realizzazione di questo studio ha richiesto l'adozione di diverse librerie. In particolare, sono utilizzate le seguenti versioni:

- Flask 2.0.x: è un micro-framework che permette la costruzione di un server semplice e veloce;
- SQLAlchemy 1.4.28: è un set completo di strumenti che permettono la manipolazioni di database;

- marshmallow 3.14.1: è una libreria indipendente per la conversione di dati complessi, come oggetti, in e da tipi di dati Python nativi.

4.2 MySQL

MySQL è il sistema più popolare di gestione di database SQL (Structured Query Language) open source sviluppato, distribuito e supportato dalla Oracle Corporation. La versione di MySQL qui utilizzata è MySQL 8.0.27.

Figura 4.2: Logo MySQL.

MySQL è un DBMS (DataBase Management System), cioè fornisce il MySQL Server in cui risiede il database. Tramite un set di strumenti MySQL permette di aggiungere, accedere ed elaborare i dati archiviati in un database. Nel capitolo precedente (Progettazione), il database progettato è formato da tabelle collegate tra loro tramite delle relazioni, motivo per cui questo MySQL è perfetto per la creazione e la gestione del database.

phpMyAdmin

phpMyAdmin è uno strumento software gratuito scritto in PHP che ha lo scopo di gestire l'amministrazione di un server di database MySQL. È possibile servirsi di un simile strumento per eseguire la maggior parte delle attività di amministrazione, inclusa la creazione di un database, l'esecuzione di query e l'aggiunta di account utente.
Tra le varie funzionalità offerte, phpMyAdmin permette di:

- Creare, sfogliare, modificare e rilasciare database, tabelle, viste, colonne e indici;

Figura 4.3: phpMyAdmin logo.

- Creare, copiare, eliminare, rinominare e modificare database, tabelle, colonne e indici;
- Aggiungere, modificare e rimuovere account utente e privilegi MySQL;

4.3 Docker

Docker è una piattaforma per sviluppatori e amministratori di sistema adatta a sviluppare, distribuire ed eseguire applicazioni su container. Inoltre, consente di separare le applicazioni in modo da poter distribuire rapidamente il software.

Figura 4.4: Logo Docker.

Mettere le applicazioni nei container apporta diversi vantaggi:

- I container Docker sono sempre portatili. Ciò significa che è possibile creare contenitori in locale e distribuirli in qualsiasi ambiente Docker (altri computer, server, cloud, ecc.);
- Sono leggeri, in quanto condividono il sistema operativo ma possono anche gestire le applicazioni più complesse;

Quando si parla dei *container*, spesso capita di confondersi con le macchine virtuali. Tuttavia, la piattaforma Docker è sempre in esecuzione sul sistema operativo. I container contengono i file binari, le librerie e l'applicazione stessa, ma non un

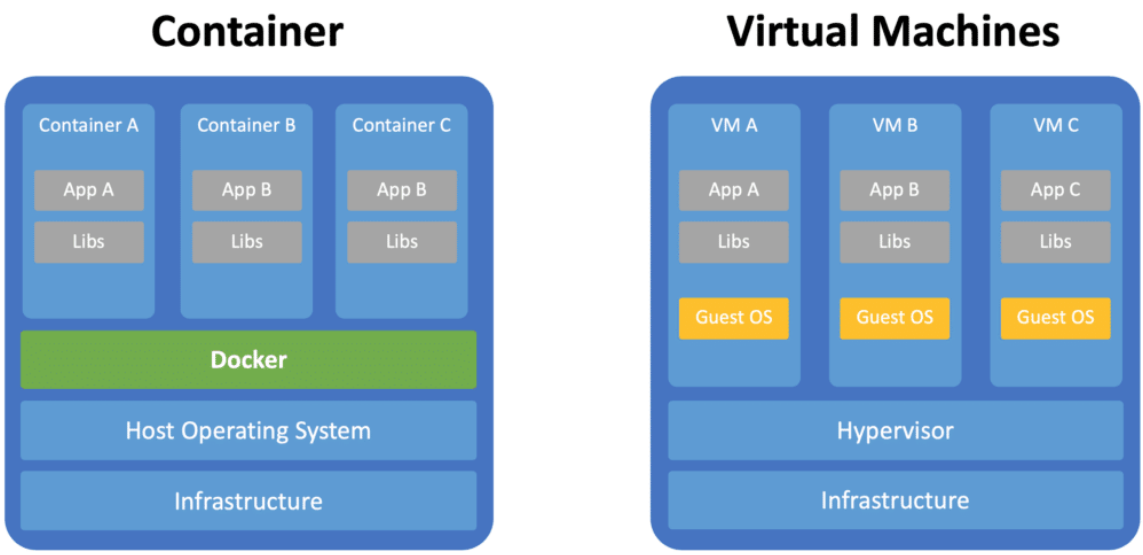

Figura 4.5: Struttura dei container e delle macchine virtuali.

sistema operativo ospite che garantisce che i container siano leggeri.
Al contrario, le macchine virtuali vengono eseguite su un hypervisor (responsabile delle macchine virtuali) e includono il proprio sistema operativo ospite. Ciò aumenta notevolmente le dimensioni delle macchine virtuali, rendendone la configurazione più complessa e richiedendo più risorse per eseguire ciascuna macchina virtuale.
Docker utilizza due elementi molto importanti: le immagini e i container.
Un'immagine Docker contiene tutto il necessario per eseguire un'applicazione come contenitore. Essa comprende:

- Codice;
- Eseguibile;
- Librerie;
- Variabili d'ambiente;
- File di configurazione.

Un'immagine può quindi essere distribuita in qualsiasi ambiente Docker e venire eseguita come container, mentre, invece un container è un'istanza in esecuzione di un'immagine. Da un'immagine è possibile creare più container su più piattaforme

Docker, ribadendo che il container, poiché viene eseguito senza la necessità di avviare un sistema operativo ospite, è leggero e limita le risorse (ad esempio la memoria) necessarie per farlo funzionare.
La versione di Docker qui utilizzata è la 20.10.12.

4.4 IPFS (InterPlanetary File System)

L'InterPlanetary File System è un protocollo di archiviazione di file distribuito che consente a una rete di computer di archiviare qualsiasi informazione in modo robusto e immutabile. Ogni computer sulla rete funge allo stesso tempo da client e da server (quindi utilizza un'architettura peer-to-peer), così da instradare e memorizzare nella cache le richieste di rete per altri client.

Figura 4.6: InterPlanetary File System logo.

IPFS ha alcune caratteristiche utili che lo fanno risaltare nello spazio crittografico, rendendolo oggi la scelta più popolare e comune per un Internet decentralizzato. Tali caratteristiche sono:

- È immutabile: i dati, una volta aggiunti alla rete, non possono più essere modificati. Gli aggiornamenti di dati già presenti nella rete possono essere pubblicati, ma questi saranno nuovi file che non andranno mai a sovrascrivere quelli vecchi;
- È a prova di duplicazione: i dati vengono suddivisi in blocchi che producono un hash diverso per ogni dato. Se due dati hanno lo stesso hash, ciò significa che sono lo stesso dato, perciò non possono esserci due hash uguali nella rete;

- È decentralizzato: anche se un gran numero di nodi va offline, l'intero sistema continua a funzionare. I tentativi di eliminare le informazioni o di censurare dei file non hanno successo fin tanto che ci sarà anche un solo nodo che contiene quella determinata informazione.

La domanda che potrebbe sorgere spontanea è: come riesce IPFS a trovare un'informazione conoscendo il suo identificativo (hash) anziché la sua posizione?
Ci sono tre principi fondamentali che danno ragione di questa peculiarità:

1. Identificazione univoca tramite indirizzamento del contenuto;
2. Collegamento dei contenuti tramite grafici aciclici diretti (DAG);
3. Scoperta dei contenuti tramite tabelle hash distribuite (DHT).

Questi tre principi si basano l'uno sull'altro per abilitare l'ecosistema IPFS.

4.4.1 Indirizzamento dei contenuti

IPFS utilizza l'indirizzamento dei contenuti per identificare il contenuto in base a ciò che contiene anziché in base alla posizione in cui si trova. Attualmente, i contenuti vengono trovati in base alla posizione mentre ogni contenuto che utilizza il protocollo IPFS ha un *identificatore di contenuto* (o CID), che è il suo valore hash.

4.4.2 Grafi aciclici diretti

IPFS e molti altri sistemi distribuiti sfruttano una struttura dati chiamata Grafi aciclici diretti (o DAG) e, in particolare, utilizzano i *DAG Merkle*, in cui ogni nodo ha un identificatore univoco che è un hash del contenuto del nodo. Ciò rimanda al concetto di indirizzamento dei contenuti. In altre parole: identificare un oggetto dati (come un nodo Merkle DAG) in base al suo valore hash vale a dire *indirizzare il contenuto*.

Per capire meglio, IPFS consente di avere un CID per ogni contenuto e permette di collegare i contenuti insieme in un DAG Merkle.

4.4.3 Tabelle hash distribuite

Per trovare quali nodi stanno ospitando un determinato contenuto, IPFS utilizza una *tabella hash distribuita* (o DHT). Una tabella hash è un database di chiavi di valori, ed è distribuita in quanto suddivisa su tutti i peer di una rete distribuita. Il progetto *libp2p* è una parte dell'ecosistema IPFS che fornisce il DHT e gestisce i peer che si connettono e parlano tra loro.
Una volta trovata la posizione di un contenuto (più precisamente, quali peer stanno memorizzando ciascuno dei blocchi che compongono il contenuto che si sta cercando), si utilizza un'altra volta il DHT per rintracciare la posizione corrente di quei peer. Quindi, per arrivare al contenuto si usa libp2p per interrogare due volte la tabella hash distribuita.
A questo punto, venuto a conoscenza dell'effettiva posizione corrente dei contenuti, si giunge al momento di connettersi a quei contenuti e ottenerli. Per richiedere e inviare blocchi ad altri peer, IPFS utilizza attualmente un modulo chiamato *Bitswap*. Esso consente di connettersi ai peer che hanno il contenuto trovato in precedenza. Bitswap invia loro una lista, dove al suo interno vengono specificati i blocchi che i peer dovranno inviare al richiedente. Una volta ricevuti questi blocchi, occorrerà eseguire l'hashing del loro contenuto per ottenere i CID e confrontarli, quindi verificarli, con i CID che sono stati richiesti.
Dopo aver definito le basi riguardanti il protocollo IPFS è necessario fornire anche la versione che verrà utilizzata nel presente lavoro, cioè la versione 0.8.0.

4.5 Hyperledger Fabric

Come viene definito nella documentazione ufficiale, "Hyperledger Fabric è una piattaforma DLT (Permissioned Distributed Ledger Technology) open source di

livello enterprise, progettata per l'uso in contesti aziendali, che offre alcune capacità chiave di differenziazione rispetto ad altre popolari piattaforme blockchain."

Figura 4.7: Hyperledger Fabric logo.

I vantaggi principali forniti da questa tecnologia sono:

- Rete permissioned: viene accordata la fiducia decentralizzata in una rete in cui i partecipanti sono noti, a differenza di una rete pubblica in cui partecipanti sono anonimi;
- Transazioni riservate: sono esposti solo i dati che si desiderano condividere con le entità del sistema;
- Architettura collegabile: è possibile personalizzare la blockchain in base alle esigenze del sistema da sviluppare;
- Iniziare con facilità: si possono programmare gli smart contract con diversi linguaggi pre-esistenti.

4.5.1 Architettura

Nell'architettura di Fabric vengono definiti alcuni concetti chiave di cui è formata questa tecnologia. Le principali componenti sono:

- Asset: consente lo scambio di tutto ciò che ha valore all'interno della rete. Gli asset sono rappresentati in Fabric come una raccolta di coppie chiave-valore;

- Registro condiviso: contiene lo stato e le proprietà di un asset. Esso, a sua volta, si compone di:

    1. Il *world state*: descrive lo stato del registro in un determinato momento. È il database del registro;
    2. La *blockchain*: è una cronologia del registro delle transazioni in cui vengono registrate tutte le transazioni;

- Smart contract: in Fabric lo smart contract è chiamato chaincode, ossia un software che definisce un asset e i metodi che permettono di modificare tale asset. Il chaincode invoca le operazioni (scrittura e lettura) di cui si ha bisogno per interagire con il registro. Il chaincode quindi è un frammento di codice che può essere scritto in Golang, Javascript o Java;

- Peer: sono elementi fondamentali nella rete in quanto essi ospitano il registro e gli smart contract. Il peer esegue il chaincode, accede ai dati del registro, approva le transazioni e si interfaccia con le applicazioni;

- Canale: i canali sono strutture logiche formate da un collezione di peer. Questa capacità consente a un gruppo di peer di creare un registro separato di transazioni;

- Organizzazioni: la rete Fabric è formata da una serie di peer che appartengono alla differenti organizzazioni partecipanti. Ogni peer ha un'identità (certificato digitale) rilasciata da un Membership Service Provider;

- Membership Service Provider (MSP): è implementata come autorità di certificazione per gestire i certificati atti a autenticare l'identità e i ruoli dei membri. Naturalmente, nessuna identità può effettuare transazioni nella rete Fabric;

- Ordering service: questo servizio impacchetta le transazioni in blocchi da consegnare ai peer su un canale. Esso garantisce la consegna delle transazioni nella rete.

4.5.2 Funzionamento

Dopo aver passato in rassegna le componenti principali di una rete Hyperledger Fabric, è conveniente vedere come queste componenti si coordinano per lavorare insieme. In questo paragrafo viene descritto cosa succede durante una normale richiesta di transazione nella rete. Poiché le principali componenti "vive" all'interno della rete sono i peer, risulta vantaggioso scendere nei dettagli di questi elementi. In precedenza, è stata data la definizione di peer però essi non sono tutti gli stessi. Esistono tre tipi differenti di peer con differenti ruoli nella rete, essi sono: gli Endorser peer, gli Anchor peer e gli Orderer peer.

Endorser peer

Dopo aver ricevuto una richiesta di transazione, questo peer si occupa di validarla, verificando i dettagli del certificato e il ruolo del richiedente. Successivamente, esegue il chaincode e simula il risultato della transazione, ma non si occupa di aggiornare lo stato del registro. Alla fine di tutte le precedenti operazioni, l'endorser peer può approvare o disapprovare la transazione. Inoltre, essendo che solo l'endorser peer è in grado di eseguire il chaincode, quest'ultimo non necessita di essere installato su tutti i nodi della rete, aumentando la scalabilità della stessa.

Anchor peer

L'anchor peer (o peer di ancoraggio) viene configurato al momento della configurazione del canale. L'anchor peer riceve gli aggiornamenti e li trasmette agli altri peer dell'organizzazione di cui fa parte. Ogni anchor peer è visibile agli altri peer così da poter essere rilevato dagli orderer peer o da altri peer.

Orderer peer

L'orderer peer è considerato il canale di comunicazione centrale per le rete Fabric. Esso è, inoltre, responsabile della coerenza del registro in tutta la rete. È anche

responsabile di creare il blocco con le transazioni e di consegnarlo a tutti i peer.

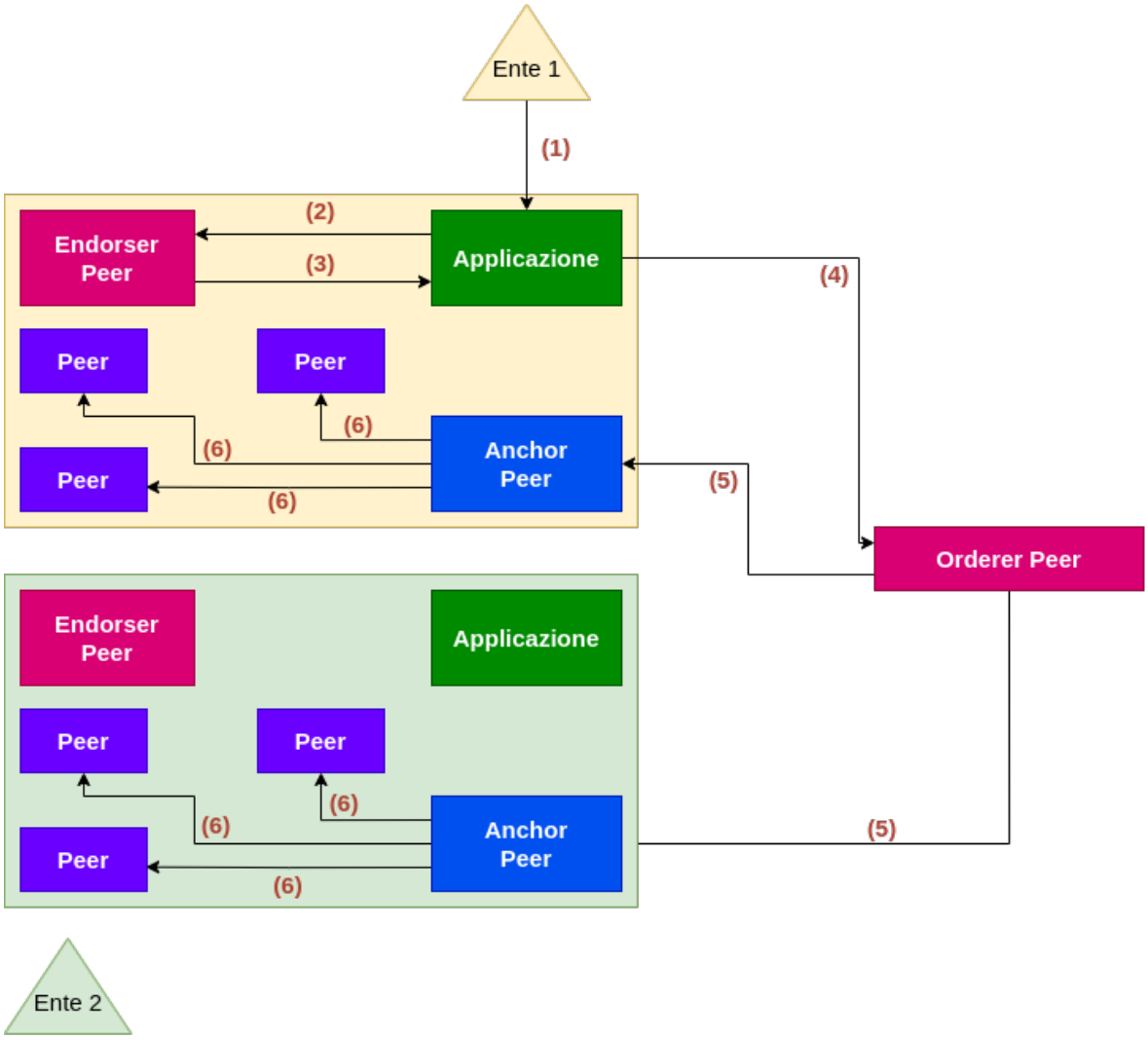

Figura 4.8: Flusso di lavoro di una transazione nella rete Fabric.

Di seguito, vengono descritti i passaggi della figura 4.8:

1. Un membro dell'Ente 1 invoca una richiesta di transazione tramite l'applicazione, cioè tramite client;

2. Il client trasmette la richiesta all'endorser peer;

3. L'endorser peer verifica i dettagli del certificato e simili, per convalidare la transazione. Quindi esegue il chaincode e restituisce il risultato (approva o disapprova la transazione) al client;

4. A questo punto, se la transazione è approvata, viene inviata all'orderer peer in modo da poter essere ordinata correttamente e inclusa in un blocco;

5. L'orderer peer include la transazione in un blocco e inoltra il blocco all'anchor peer delle organizzazioni della rete Fabric;

6. Infine, gli anchor peer trasmettono il blocco agli altri peer dell'organizzazione. Questi singoli peer aggiornano, quindi, il registro locale con l'ultimo blocco. Così tutta la rete ottiene la sincronizzazione del registro condiviso.

Infine, riguardo alla versione di Hyperledger Fabric, quella che viene utilizzata è la versione 2.2.

CAPITOLO 5

Implementazione

Nel seguente capitolo, vengono applicate le tecnologie approfondite in precedenza, mettendo ognuna di esse in correlazione basandosi sulle linee guida fornite dalla fase di progettazione.

5.1 Gestione documenti

L'operazione essenziale di questo studio è la gestione dei documenti (titoli) di ogni entità facente parte di un istituto accademico. A tal fine, si ritiene necessario realizzare un diagramma di sequenza che possa dare un'idea ben precisa del flusso di inserimento di un documento all'interno del sistema. In particolare, l'operazione di assegnazione del titolo viene lasciata esclusivamente ai promotori degli enti connessi alla blockchain. Il diagramma di sequenza fornisce tutti i passaggi per l'inserimento di un titolo. Per prima cosa, il promotore deve accedere alla piattaforma per verificare chi effettivamente vuole compiere l'operazione. Una volta effettuato l'accesso, il promotore inserisce tramite la dashboard (lato client) tutti i dati richiesti per l'avvio dell'operazione ed effettua una richiesta al server, utilizzando una richiesta HTTP

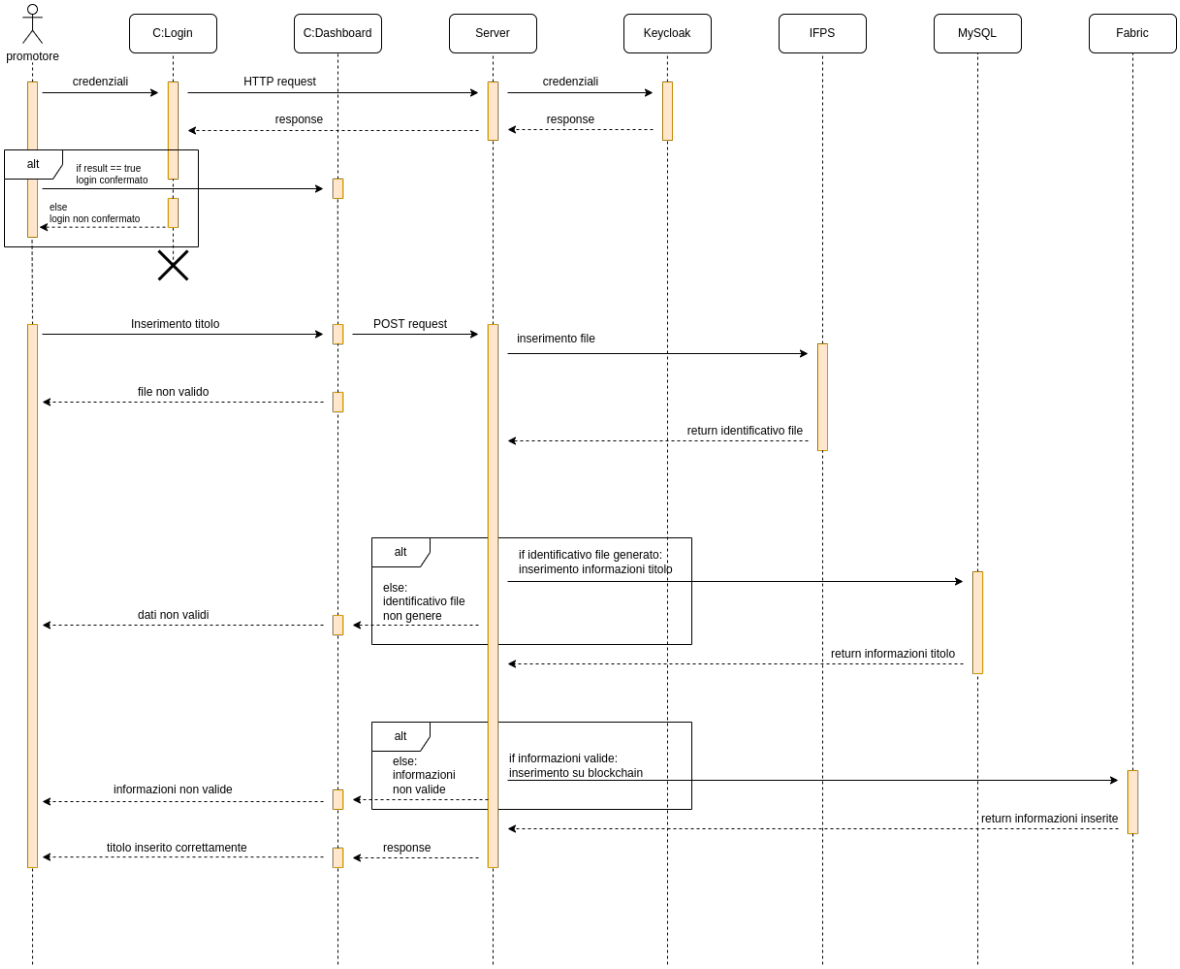

Figura 5.1: Diagramma di sequenza: inserimento titolo

(HyperText Transfer Protocol), nello specifico una POST request per soddisfare uno dei principi delle architetture REST (REpresentational State Transfer). A questo punto, il server si occuperà di caricare su IPFS (InterPlanetar File System) il file del documento di un determinato utente. Se il file è valido, sarà restituito l'identificativo hash associato al file, altrimenti sarà generato un errore. Supponendo che l'operazione di inserimento sia andata a buon fine, è possibile procedere con l'inserimento di tutte le informazioni riguardanti il titolo sul database. Se il procedimento giunge a completamento senza intoppi, si passa all'ultima operazione, ossia l'inserimento del titolo all'interno della blockchain. I dati inseriti saranno l'id dell'utente a cui viene assegnato il titolo, il nome del titolo, l'id del database del titolo e l'hash identificativo generato da IPFS. Questa operazione sulla blockchain produce un hash che identifica univocamente la transazione appena avvenuta.

5.2 Blockchain

Per l'implementazione della blockchain viene utilizzata la test-network fornita con Hyperledger Fabric. In particolare, la test-network è una semplice implementazione di una blockchain i cui partecipanti sono solamente due. Diversamente, durante la fase di progettazione (Capitolo 3) sono stati definiti tre partecipanti alla rete. A tal proposito, si dovranno dunque apportare delle modifiche al codice sorgente di Fabric per implementare la blockchain in base alle specifiche definite in precedenza.

5.2.1 Aggiunta organizzazione

Gli attori che interagiranno con la blockchain sono tre: l'università e due enti. Poiché la test-network, al primo avvio, supporta soltanto l'utilizzo di due attori, deve essere modificato il codice sorgente per aggiungere un nuovo utente. Per far ciò, bisogna appendere il seguente frammento di codice all'interno del file *configtx.yaml* presente dentro la directory *configtx*.

```
Organizations:
    - &Org3
        Name: Org3MSP
        ID: Org3MSP
        MSPDir: ../organizations/peerOrganizations/org3.example.com/msp
        Policies:
            Readers:
                Type: Signature
                Rule: "OR('Org3MSP.admin', 'Org3MSP.peer', 'Org3MSP.client')"
            Writers:
                Type: Signature
                Rule: "OR('Org3MSP.admin', 'Org3MSP.client')"
            Admins:
                Type: Signature
                Rule: "OR('Org3MSP.admin')"
            Endorsement:
```

```
                Type: Signature
                Rule: "OR('Org3MSP.peer')"
```

Listing 5.1: Aggiunta di una terza organizzazione.

Una volta aggiunto il terzo partecipante, bisogna modificare i tre file contenuti all'interno della directory *docker*. Nel file *docker-compose-ca.yaml* viene fissata la seguente parte di codice.

```
ca_org3:
    image: hyperledger/fabric-ca:latest
    labels:
      service: hyperledger-fabric
    environment:
      - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
      - FABRIC_CA_SERVER_CA_NAME=ca-org3
      - FABRIC_CA_SERVER_TLS_ENABLED=true
      - FABRIC_CA_SERVER_PORT=11054
      - FABRIC_CA_SERVER_OPERATIONS_LISTENADDRESS=0.0.0.0:20054
    ports:
      - "11054:11054"
      - "20054:20054"
    command: sh -c 'fabric-ca-server start -b admin:adminpw -d'
    volumes:
      - ../organizations/fabric-ca/org3:/etc/hyperledger/fabric-ca-server
    container_name: ca_org3
    networks:
      - test
```

Listing 5.2: Configurazione della Certificate Authority per l'organizzazione 3

In questo frammento viene generata la certificazione che servirà ad ogni partecipante alla rete per autenticarsi.

Successivamente, viene modificato il file *docker-compose-couch.yaml* nel quale viene attaccata la seguente porzione di codice.

```
couchdb4:
```

```
    container_name: couchdb4
    image: couchdb:3.1.1
    labels:
      service: hyperledger-fabric
    environment:
      - COUCHDB_USER=admin
      - COUCHDB_PASSWORD=adminpw
    ports:
      - "9984:5984"
    networks:
      - test

  peer0.org3.example.com:
    environment:
      - CORE_LEDGER_STATE_STATEDATABASE=CouchDB
      - CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=couchdb4:5984
      - CORE_LEDGER_STATE_COUCHDBCONFIG_USERNAME=admin
      - CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD=adminpw
    depends_on:
      - couchdb4
    networks:
      - test
```

Listing 5.3: Configurazione del database associato all'organizzazione 3

In quest'ultima porzione viene creato un nuovo container che fornisce un'implementazione di CouchDB. Inoltre, si imposta l'organizzazione tre (cioè il terzo partecipante aggiunto) per accedere a tale database.

Infine, rimane l'ultimo file, cioè *docker-compose-test-net.yaml*.

```
peer0.org3.example.com:
    container_name: peer0.org3.example.com
    image: hyperledger/fabric-peer:latest
    labels:
      service: hyperledger-fabric
    environment:
```

```
      #Generic peer variables
      - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
      - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=fabric_test
      - FABRIC_LOGGING_SPEC=INFO
      #- FABRIC_LOGGING_SPEC=DEBUG
      - CORE_PEER_TLS_ENABLED=true
      - CORE_PEER_PROFILE_ENABLED=false
      - CORE_PEER_TLS_CERT_FILE=/etc/hyperledger/fabric/tls/server.crt
      - CORE_PEER_TLS_KEY_FILE=/etc/hyperledger/fabric/tls/server.key
      - CORE_PEER_TLS_ROOTCERT_FILE=/etc/hyperledger/fabric/tls/ca.crt
      # Peer specific variables
      - CORE_PEER_ID=peer0.org3.example.com
      - CORE_PEER_ADDRESS=peer0.org3.example.com:11051
      - CORE_PEER_LISTENADDRESS=0.0.0.0:11051
      - CORE_PEER_CHAINCODEADDRESS=peer0.org3.example.com:11052
      - CORE_PEER_CHAINCODELISTENADDRESS=0.0.0.0:11052
      - CORE_PEER_GOSSIP_BOOTSTRAP=peer0.org3.example.com:11051
      - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer0.org3.example.com:11051
      - CORE_PEER_LOCALMSPID=Org3MSP
      - CORE_OPERATIONS_LISTENADDRESS=peer0.org3.example.com:9447
      - CORE_METRICS_PROVIDER=prometheus
    volumes:
        - ${DOCKER_SOCK}:/host/var/run/docker.sock
        - ../organizations/peerOrganizations/org3.example.com/peers/peer0
    .org3.example.com/msp:/etc/hyperledger/fabric/msp
        - ../organizations/peerOrganizations/org3.example.com/peers/peer0
    .org3.example.com/tls:/etc/hyperledger/fabric/tls
        - peer0.org3.example.com:/var/hyperledger/production
    working_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer
    command: peer node start
    ports:
      - 11051:11051
      - 9447:9447
    networks:
```

```
39      - test
```

Listing 5.4: Configurazione del container docker per l'organizzazione 3

Dopo aver aggiunto il frammento soprascritto, si avrà una configurazione completa di una blockchain Fabric, che contiene tre diversi partecipanti (o organizzazioni). Quando verrà mandato in esecuzione il comando

```
1 ./network.sh up
```

saranno generati ed eseguiti diversi container, tra cui:

- Il container per l'orderer peer, che come detto in precedenza si occupa di mantenere il registro della blockchain coerente;
- Il container contenente il peer dell'università;
- Il container contenente il peer per il primo ente;
- Il container contenente il peer per il secondo ente.

Per semplificare l'aggiunta di un nuovo partecipante e per rendere minime le modifiche al codice sorgente, è stato deciso di non modificare il nome delle organizzazioni, bensì di mantenere le etichette definite da Hyperledger, cioè Org1, Org2 e Org3 che sono, rispettivamente, l'università, l'ente 1 e l'ente 2.

5.2.2 Creazione canali

Spetta, ora, la creazione di due canali, che servono a mettere in comunicazione ogni ente con l'università. La comunicazione può avvenire soltanto tra università e ente, quindi si avrà:

- canale1: nel quale potranno scrivere l'università e l'ente 1;
- canale2: nel quale potranno scrivere l'università e l'ente 2.

Per la creazione dei canali, Fabric ha uno script. Esso è il file *createChannel.sh*. Da riga di comando viene eseguito:

```
./network.sh createChannel -c canale1
```

Listing 5.5: Comando per la creazione del primo canale

Stessa operazione vale per il canale2 soltanto che si dovrà effettuare una modifica allo script *createChannel.sh* presente nella directory *script*. All'interno dello script ci sarà il frammento di codice sottostante.

```
infoln "Joining org1 peer to the channel..."
joinChannel $ORG1
infoln "Joining org2 peer to the channel..."
joinChannel $ORG2
infoln "Setting anchor peer for org1..."
setAnchorPeer $ORG1
infoln "Setting anchor peer for org2..."
setAnchorPeer $ORG2
```

Listing 5.6: Modifica per l'unione delle organizzazioni al canale

La modifica da effettuare è quella di sostituire tutte le occorrenze presenti di *ORG2* con *ORG3*. Una volta eseguito il comando sottostante sarà completata l'operazione di unione di Org1 e Org3 al canale 2.

```
./network.sh createChannel -c canale2
```

Listing 5.7: Comando per la creazione del secondo canale

5.2.3 Deployment chaincode

Arrivati a questo punto, la blockchain è funzionante, sebbene allo stato attuale non sia stato installato alcun chaincode (Smart Contract) sui canali, di conseguenza non è possibile far cambiare lo stato globale e non ci sono operazioni definite da poter utilizzare. Il seguente script è la definizione del chaincode sviluppato in Javascript.

```
'use strict';

const { Contract } = require('fabric-contract-api');

```

```
class Titolo extends Contract {
    //Inserisce un nuovo titolo allo stato della blockchain
    async insertTitolo(ctx, new_id_titolo, new_nome_titolo,
    new_id_studente, new_ipfs_hash) {
        const exists = await this.titoloExists(ctx, new_id_titolo);
        if (exists) {
            throw new Error(`Il titolo ${new_id_titolo} esiste gia'.`);
        }
        const titolo = {
            nome_titolo: new_nome_titolo,
            id_studente: new_id_studente,
            ipfs_hash: new_ipfs_hash,
        };
        await ctx.stub.putState(new_id_titolo.toString(), Buffer.from(
    JSON.stringify(titolo)));
        return ctx.stub.getTxID(); //Restituisce l'id hash della
    transazione
    }

    //Restituisce un titolo che ha come id quello preso in input
    async readTitolo(ctx, id_titolo) {
        const titoloJSON = await ctx.stub.getState(id_titolo);
        if(!titoloJSON || titoloJSON.length === 0) {
            throw new Error(`Il titolo ${id_titolo} non esiste.`);
        }
        return titoloJSON.toString(); //Restituisce il titolo in formato
    JSON convertito in stringa.
    }

    //Elimina il titolo con l'id preso in input
    async deleteTitolo(ctx, id_titolo) {
        const exists = await this.titoloExists(ctx, id_titolo);
        if(!exists) {
            throw new Error (`Il titolo ${id_titolo} non esiste`);
```

```
35          }
36          return ctx.stub.deleteState(id_titolo);
37      }
38
39      //Verifica se un titolo e' gia' presente nella blockchian. E' un
    metodo molto importante come supporto per gli altri metodi.
40      async titoloExists(ctx, id_titolo) {
41          const titoloJSON = await ctx.stub.getState(id_titolo);
42          return titoloJSON && titoloJSON.length > 0;
43      }
44
45      //Restituisce tutti i titoli presenti nella blockchain
46      async getAllTitoli(ctx) {
47          const allResults = [];
48          const iterator = await ctx.stub.getStateByRange('', '');  //
    Prende in input il range nel quale prendere i titoli. Non viene
    passato alcun parametro poiche' vogliamo farci restituire tutti i
    titoli.
49          let result = await iterator.next();
50          while(!result.done) {
51              const strValue = Buffer.from(result.value.value.toString()).
    toString('utf8');
52              let record;
53              try {
54                  record = JSON.parse(strValue);
55              } catch(err) {
56                  record = strValue;
57              }
58              allResults.push(record);
59              result = await iterator.next();
60          }
61          return JSON.stringify(allResults);
62      }
63  }
```

```
64
65 module.exports = Titolo;
```

Listing 5.8: Chaincode per le operazioni sul titolo nella blockchain.

Nel chaincode sono state definite le principali operazioni che possono essere effettuate sulla blockchain. Ad esempio, il metodo *insertTitolo* prende in input quattro parametri: l'identificativo del titolo, il nome, l'identificativo dell'utente a cui appartiene il titolo e l'hash del file caricato su IPFS (InterPlanetary File System). Successivamente, il metodo verifica che il titolo non sia già presente nella blockchain. Se questa eventualità è accertata allora i dati presi in input verranno inseriti nello stato della blockchain, cioè immagazzinati al suo interno.
Definite le regole e le operazioni a cui la blockchain dovrà sottostare, è possibile passare alla fase di composizione del server e della RESTful API.

5.3 Server

Ogni operazione che avviene sulla piattaforma genera una richiesta al server. All'interno del server, per gestire e comunicare con tutte le tecnologie presenti, si utilizza un'API (Application Programming Interface), quindi il primo obiettivo è l'implementazione di un'API. Questo permette, tramite richieste HTTP (HyperText Transfer Protocol), di soddisfare tutte le operazioni a disposizione per gli utenti. Nello specifico, viene implementata una soluzione REST (REpresentional State Transfer) che consiste di cinque principi "architetturali":

1. Identificazione delle risorse: ogni risorsa deve essere identificata univocamente attraverso un URI (Uniform Resource Identifier);

2. Utilizzo esplicito dei metodi HTTP: ogni metodo definito nel protocollo HTTP deve essere mappato con le operazioni CRUD (Create, Read, Update e Delete);

3. Risorse autodescrittive: ogni risultato elaborato deve utilizzare un formato dati generico, come XML o JSON;

4. Collegamenti tra risorse: tutto quello che si deve sapere su una risorsa, deve essere contenuto nella risposta (quindi dall'HTTP response);

5. Comunicazione senza stato: realizza il principio di comunicazione *stateless*, ove ogni richiesta effettuata all'API non ha alcuna relazione con le precedenti richieste, né con quelle successive.

Per lo sviluppo del server e dell'API si impiega il framework Flask. Questo, attraverso un semplice file Python, permette di creare velocemente la base per il server. Attraverso l'uso di alcuni costrutti forniti da questo framework è possibile riportare una maggiore modularità. Per migliorare la modularità del sistema, Flask mette a disposizione lo strumento *blueprint*. I blueprint permettono di semplificare notevolmente il funzionamento di una applicazione di grandi dimensioni e rendono abbastanza semplice estendere le funzionalità dell'applicazione. L'utilizzo che ne viene fatto nel presente lavoro è quello di *registrare* un prefisso URL (quindi una *route*), dal quale l'API fornirà delle operazioni a un oggetto blueprint.
Oltre alla funzione di blueprint, vi è anche la presenza di una libreria chiamata *Marshmallow*. Come viene riportato dalla documentazione ufficiale, "Marshmallow è una libreria che supporta la conversione di tipi di dati complessi, come oggetti, in e da tipi di dati Python nativi": vale a dire che permette di creare degli schemi per convertire un oggetto, ad esempio un oggetto che rappresenta una tabella di un database.

```
1  from config_app import app
2  from config_db import db
3  from blueprints.studente import studente_bp
4  from blueprints.corso import corso_bp
5  from blueprints.personale import personale_bp
6  from blueprints.categoria import categoria_bp
7  from blueprints.titolo import titolo_bp
8  from blueprints.evento import evento_bp
9  from blueprints.tipologia import tipologia_bp
10 from blueprints.partecipazione import partecipazione_bp
```

```
11 from blueprints.dipartimento import dipartimento_bp
12 from blueprints.universita import universita_bp
13 from blueprints.citta import citta_bp
14 from blueprints.nazione import nazione_bp
15 from blueprints.promotore import promotore_bp
16 from blueprints.ente import ente_bp
17
18 app.register_blueprint(studente_bp)
19 app.register_blueprint(corso_bp)
20 app.register_blueprint(personale_bp)
21 app.register_blueprint(categoria_bp)
22 app.register_blueprint(titolo_bp)
23 app.register_blueprint(evento_bp)
24 app.register_blueprint(ente_bp)
25 app.register_blueprint(tipologia_bp)
26 app.register_blueprint(partecipazione_bp)
27 app.register_blueprint(dipartimento_bp)
28 app.register_blueprint(universita_bp)
29 app.register_blueprint(citta_bp)
30 app.register_blueprint(nazione_bp)
31 app.register_blueprint(promotore_bp)
32
33 if __name__ == '__main__':
34     db.create_all()
35     app.run(debug=True)
```

Listing 5.9: Main file del server.

Il file *app.py* è il file principale del server. I primi due elementi *import* sono file di configurazione per il database e per il server stesso.

```
1 from flask import Flask
2
3 app = Flask(__name__)
4 app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql+pymysql://root@localhost/
     tesi'
```

```
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
```

Listing 5.10: File di configurazione di Flask.

```
from config_app import app
from flask_sqlalchemy import SQLAlchemy

db = SQLAlchemy(app)
```

Listing 5.11: File di configurazione per il database.

In base a quanto visto in precedenza, l'operazione più importante per il presente studio è la gestione dei titoli. A tal proposito, vi sono tre script (model, blueprint e schema) principali che implementano le possibili operazioni sui titoli.

```
from flask import Flask, json
from sqlalchemy.orm import query
from config_db import db

#La classe Titolo contiene la struttura che ha la tabella dentro il
    database
class Titolo(db.Model):
    __tablename__ = 'titolo'
    id_titolo = db.Column(db.Integer, primary_key=True)
    id_evento = db.Column(db.Integer, db.ForeignKey('evento.id_evento'),
    unique=False, nullable=True)
    id_personale = db.Column(db.Integer, db.ForeignKey('personale.
    id_personale'), unique=False, nullable=True)
    id_studente = db.Column(db.Integer, db.ForeignKey('studente.
    id_studente'), unique=False, nullable=True)
    nome = db.Column(db.String(100), unique=False, nullable=False)
    ipfs_hash = db.Column(db.String(256), unique=True, nullable=False)
    fabric_hash = db.Column(db.String(256), unique=True, nullable=False)

    evento = db.relationship("Evento")
    personale = db.relationship("Personale")
    studente = db.relationship("Studente")
```

```
19
20      def __init__(self, id_evento, id_personale, id_studente, nome,
        ipfs_hash, fabric_hash):
21          self.id_evento = id_evento
22          self.id_personale = id_personale
23          self.id_studente = id_studente
24          self.nome = nome
25          self.ipfs_hash = ipfs_hash
26          self.fabric_hash = fabric_hash
```

Listing 5.12: Model per il titolo.

Il file titolo_model è la definizione della tabella Titolo e di tutte le operazioni possibili.

```
1  from config import *
2  from flask_marshmallow import Marshmallow
3
4  ma = Marshmallow(app)
5
6  class TitoloSchema(ma.Schema):
7      class Meta:
8          fields = (
9              'id_titolo',
10             'id_evento',
11             'id_personale',
12             'id_studente',
13             'nome',
14             'ipfs_hash',
15             'fabric_hash'
16         )
17
18 titolo_schema = TitoloSchema()
19 titoli_schema = TitoloSchema(many=True)
```

Listing 5.13: File di definizione dello schema.

```
1  import json
```

```
2  from flask import Blueprint, request, session
3  from flask.json import jsonify
4  from blockchain import Blockchain
5  from config_db import db
6  from config_ipfs import client
7  from models.titolo_model import Titolo, Informazione
8  from schemas.titolo_schema import titolo_schema, titoli_schema,
       informazione_schema
9
10 titolo_bp = Blueprint('titolo_bp', __name__)
11
12 @titolo_bp.route("/titolo", methods=["GET"])
13 def get_titoli():
14     all_titoli = Titolo.query.all()
15     result = titoli_schema.dump(all_titoli)
16     return jsonify(result)
17
18 @titolo_bp.route("/titolo", methods=["POST"])
19 def add_titolo():
20     data = json.loads(request.form.get('data'))
21     id_promotore = data['id_promotore']
22     id_evento = data['id_evento']
23     id_personale = data['id_personale']
24     id_studente = data['id_studente']
25     nome = data['nome']
26     file = request.files.get('titolo')
27     ipfs_hash = client.add(file)['Hash']
28     descrizione = data['descrizione']
29     data_rilascio = data['data_rilascio']
30     new_informazione = Informazione(descrizione, data_rilascio)
31     new_titolo = Titolo(id_evento, id_personale, id_studente, nome,
       ipfs_hash)
32     db.session.add(new_informazione)
33     db.session.add(new_titolo)
```

```
34      db.session.commit()
35      last_titolo = db.session.query(Titolo).order_by(db.desc(Titolo.
    id_titolo)).first()
36      last_id_titolo = last_titolo.getIdTitolo()
37      if(id_promotore == 1):
38          b = Blockchain(1)
39      else:
40          b = Blockchain(2)
41      if(id_studente is not None):
42          return_value = b.insertTitolo(last_id_titolo, nome, id_studente,
    ipfs_hash)
43      else:
44          return_value = b.insertTitolo(last_id_titolo, nome, id_personale,
     ipfs_hash)
45      if(return_value == '500'):
46          return {"error": "500"}
47      titolo = Titolo.query.get(last_id_titolo)
48      titolo.fabric_hash = return_value
49      db.session.commit()
50      return titolo_schema.jsonify(new_titolo)
51
52  @titolo_bp.route("/titolo/<id>", methods=["GET"])
53  def get_titolo(id):
54      titolo = Titolo.query.get(id)
55      return titolo_schema.jsonify(titolo)
56
57  @titolo_bp.route("/titolo/<id>", methods=["DELETE"])
58  def delete_titolo(id):
59      titolo = Titolo.query.get(id)
60      info = Informazione.query.get(id)
61      db.session.delete(info)
62      db.session.delete(titolo)
63      db.session.commit()
64      return titolo_schema.jsonify(titolo)
```

```
65
66 @titolo_bp.route("/titolo/informazione/<id>", methods=['GET'])
67 def get_informazione(id):
68     info = Informazione.query.get(id)
69     return informazione_schema.jsonify(info)
```

Listing 5.14: Blueprint per il titoli.

È molto importante notare che per la funzione di inserimento è definito anche un identificativo per il promotore (un dipendente di uno degli due enti) che è indicato per impostare le variabili d'ambiente operando per un determinato ente definito nell'architettura della blockchain.

In seguito, deve essere implementato uno script che permetta la comunicazione tra server e blockchain. Lo script *blockchain.py*, tramite i moduli Python *subprocess* e *os*, consente di eseguire dei comandi direttamente sulla BASH.

```
1  import subprocess, os
2  import json
3
4  my_env = dict(os.environ)
5
6  class Blockchain():
7      def __init__(self, promotore):
8          my_env['PATH'] = "/home/bernardo/go/src/github.com/
       BernardoDePietro/fabric-samples/unime/api/../../bin:$PATH"
9          my_env['FABRIC_CFG_PATH'] = "/home/bernardo/go/src/github.com/
       BernardoDePietro/fabric-samples/unime/api/../../config/"
10         my_env['CORE_PEER_TLS_ENABLED'] = "true"
11         if(promotore == 1):
12             my_env['CORE_PEER_LOCALMSPID'] = "Org2MSP"
13             my_env['CORE_PEER_TLS_ROOTCERT_FILE'] = "/home/bernardo/go/
       src/github.com/BernardoDePietro/fabric-samples/unime/api/../
       organizations/peerOrganizations/org2.example.com/peers/peer0.org2.
       example.com/tls/ca.crt"
14             my_env['CORE_PEER_MSPCONFIGPATH'] = "/home/bernardo/go/src/
```

```
github.com/BernardoDePietro/fabric-samples/unime/api/../organizations/
peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp"
15          my_env['CORE_PEER_ADDRESS'] = "localhost:9051"
16          canale = "canale1"
17      else:
18          my_env['CORE_PEER_LOCALMSPID'] = "Org3MSP"
19          my_env['CORE_PEER_TLS_ROOTCERT_FILE'] = "/home/bernardo/go/
src/github.com/BernardoDePietro/fabric-samples/unime/api/../
organizations/peerOrganizations/org3.example.com/peers/peer0.org3.
example.com/tls/ca.crt"
20          my_env['CORE_PEER_MSPCONFIGPATH'] = "/home/bernardo/go/src/
github.com/BernardoDePietro/fabric-samples/unime/api/../organizations/
peerOrganizations/org3.example.com/users/Admin@org3.example.com/msp"
21          my_env['CORE_PEER_ADDRESS'] = "localhost:11051"
22          canale = "canale2"
23
24  def insertTitolo(self, id_titolo, nome_titolo, id_studente, ipfs_hash
):
25      if(my_env["CORE_PEER_LOCALMSPID"] == "Org2MSP"):
26          create_request = "peer chaincode invoke -o localhost:7050 --
ordererTLSHostnameOverride orderer.example.com --tls --cafile ${PWD
}/../organizations/ordererOrganizations/example.com/orderers/orderer.
example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C canale1 -n
titolo --peerAddresses localhost:7051 --tlsRootCertFiles ${PWD}/../
organizations/peerOrganizations/org1.example.com/peers/peer0.org1.
example.com/tls/ca.crt --peerAddresses localhost:9051 --
tlsRootCertFiles ${PWD}/../organizations/peerOrganizations/org2.
example.com/peers/peer0.org2.example.com/tls/ca.crt -c '{\"function
\":\"insertTitolo\",\"Args\":[\"" + str(id_titolo) + "\", \"" + str(
nome_titolo) + "\", \"" + str(id_studente) + "\", \"" + str(ipfs_hash)
 + "\"]}'"
27      else:
28          create_request = "peer chaincode invoke -o localhost:7050 --
ordererTLSHostnameOverride orderer.example.com --tls --cafile ${PWD
```

```
}/../organizations/ordererOrganizations/example.com/orderers/orderer.
example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C canale2 -n
titolo --peerAddresses localhost:7051 --tlsRootCertFiles ${PWD}/../
organizations/peerOrganizations/org1.example.com/peers/peer0.org1.
example.com/tls/ca.crt --peerAddresses localhost:11051 --
tlsRootCertFiles ${PWD}/../organizations/peerOrganizations/org3.
example.com/peers/peer0.org3.example.com/tls/ca.crt -c '{\"function
\":\"insertTitolo\",\"Args\":[\"" + str(id_titolo) + "\", \"" + str(
nome_titolo) + "\", \"" + str(id_studente) + "\", \"" + str(ipfs_hash)
 + "\"]}'"
29      output = subprocess.run(create_request, shell=True, env=my_env,
capture_output=True)
30      status, payload = self.getResponse(output.stderr.decode("utf-8"))
31      if(status == '200'):
32          return payload
33      return status
34
35  def getTitolo(self, id_titolo):
36      read_request = "peer chaincode query -C " + str(self.canale) + "
-n titolo -c '{\"Args\":[\"readTitolo\", \"" + str(id_titolo) + "\"]}'
"
37      titolo = subprocess.run(read_request, shell=True, env=my_env,
capture_output=True)
38      return json.loads(titolo.stderr.decode('utf-8'))
39
40  def deleteTitolo(self, id_titolo):
41      delete_request = "peer chaincode query -C " + str(self.canale) +
" -n titolo -c '{\"Args\":[\"deleteTitolo\", \"" + str(id_titolo) + "
\"]}'"
42      titolo = subprocess.run(delete_request, shell=True, env=my_env,
capture_output=True)
43      return json.loads(titolo.stderr.decode('utf-8'))
44
45  def getAllTitoli(self):
```

```
46        read_request = "peer chaincode query -C " + str(self.canale) + "
  -n titolo -c '{\"Args\":[\"getAllTitoli\"]}'"
47        titoli = subprocess.run(read_request, shell=True, env=my_env,
  capture_output=True)
48        return json.loads(titoli.stderr.decode('utf-8'))
49
50    def getResponse(input):
51        temp = ''
52        status = ''
53        payload = ''
54        for char in input:
55            char = str(char)
56            if(temp.startswith('status:')):
57                if(char.isdigit()):
58                    status += char
59            if(temp.startswith('payload:\"')):
60                if(char != '\"'):
61                    payload += char
62            if(char != ' '):
63                temp += char
64            else:
65                temp = ''
66        return status, payload
```

Listing 5.15: Classe Blockchain contenente le operazioni possibili.

Analizzando il codice soprascritto, si nota che:

- È stato definito un dizionario per gestire le variabili d'ambiente. Queste sono inserite nel processo *run*;
- È stata implementata una classe chiamata Blockchain;
- La classe Blockchain ha un metodo costruttore che prende in input l'identificativo del promotore e imposta le variabili d'ambiente in riferimento all'ente di cui fa parte;

- Tramite il metodo *run* del modulo *subprocess* viene eseguita la richiesta alla blockchain;

- Poiché non esiste nessuna funzione valida per gestire l'output, che si presenta in forma di stringa sulla shell, è stata implementata una soluzione tramite il metodo *getResponse*. Questo metodo cattura l'input prodotto sulla shell e seleziona lo status (cioè lo stato generato dall'operazione) e il payload, che sarebbe l'identificativo hash della transazione.

5.3.1 IPFS

In merito a IPFS, dopo averlo installato, basta utilizzare la libreria Python *ipfshttpclient*. Mediante un file di configurazione è possibile collegarsi a IPFS. Lo script è il seguente:

```
1 import ipfshttpclient
2 client = ipfshttpclient.connect()
```

Listing 5.16: Configurazione di IPFS.

Ogni qualvolta si vorrà prendere o inserire un documento su IPFS basterà importare l'oggetto *client*. Quest'oggetto metterà a disposizione dell'utente diversi metodi per comunicare con IPFS. Un esempio è il metodo *add* utilizzato in precedenza nello script *titolo.py*, dove esso rende possibile l'aggiunta di un file ricevuto in input e restituisce l'hash per accedere successivamente a tale risorsa.

CAPITOLO 6

Test sperimentali

Nel capitolo seguente, è necessario evidenziare tutte le caratteristiche (hardware e software) che compongono il sistema da cui sono effettuati i test. I test sono eseguiti sulle tecnologie principali del software sviluppato, quindi: MySQL, InterPlanetary File System e Hyperledger Fabric. Il primo test è effettuato sulla base dei dati prodotti dall'operazione di inserimento, in particolare la media di ogni operazione e il tempo totale che il test stesso impiega a completare tutte le operazioni. Nel secondo test è eseguito su due diverse implementazioni di Hyperledger Fabric.

6.1 Caratteristiche sistema

Risulta utile quando si effettuano dei test specificare le caratteristiche del sistema da dove essi vengono eseguiti. Per la configurazione hardware, si hanno a disposizione le seguenti componenti:

- Processore: Intel Core i5-10210U @ 1.60GHz Quad-Core;
- RAM (Random Access Memory): 8 GB;

- SSD (Solid State Disk): 91,3 GB.

La configurazione software è invece composta da:

- Sistema operativo: Elementary OS 6.1 Jólnir basato su Ubuntu 20.04.3 LTS;
- Python: versione 3.8.10;
- InterPlanetary File System: versione 0.8.0;
- MySQL: versione 8.0.28;
- Hyperledger Fabric: versione 2.2.

6.2 Primo test

Il primo test è quello che coinvolge più tecnologie da valutare: IPFS, MySQL e Fabric. Nello svolgere l'indagine, è stato considerato il tempo d'esecuzione medio per ogni operazione di inserimento e il tempo totale impiegato per completare l'intera operazione. Per verificare come si comportano le diverse tecnologie al variare delle operazioni, tutti i test sono stati divisi in 10, 100 e 1000 operazioni.
Nella prima tabella, i dati rappresentano il tempo d'esecuzione medio per ogni transazione. Tramite il grafico (Figura 6.1) è possibile notare che all'aumentare delle operazioni il tempo d'esecuzione medio decresce in maniera impercettibile.

N° Operazioni	IPFS	MySQL	Fabric
10	3,46	2,20	105,55
100	3,42	2,05	106,40
1000	3,32	2,01	98,85

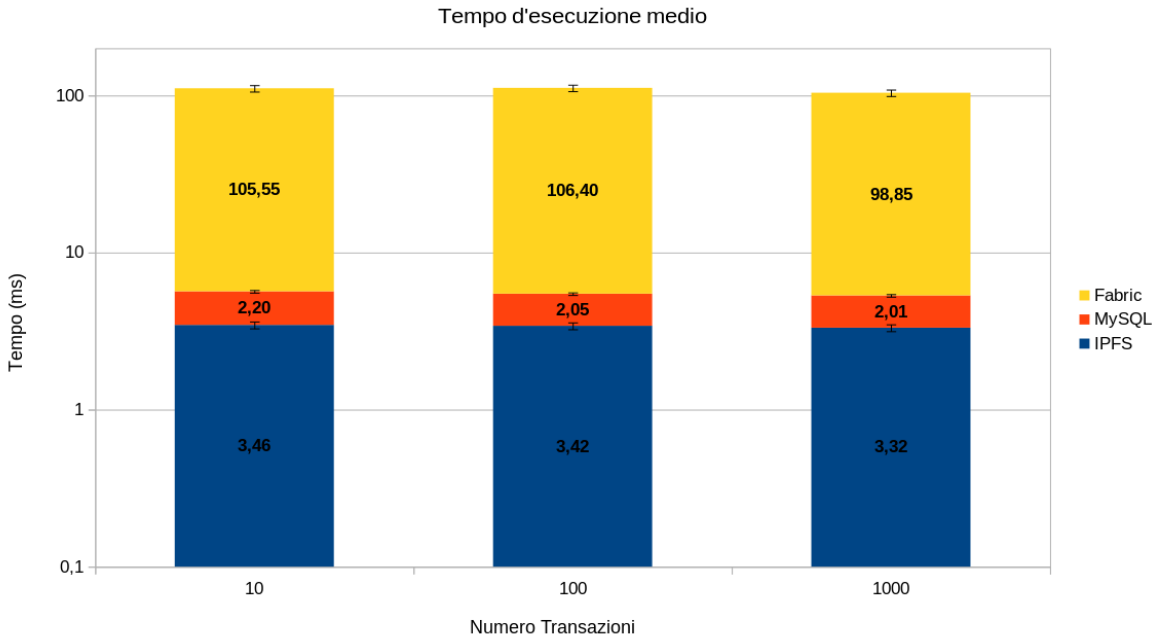

Figura 6.1: Rappresentazione e confronto del d'esecuzione medio al crescere delle operazioni.

Nella seconda tabella, i dati riproducono il tempo d'esecuzione totale per il completamento di tutte le operazioni. In maniera evidente nel grafico (Figura 6.2) si nota che al crescere del numero di operazione da eseguire cresce linearmente anche il tempo d'esecuzione totale.

N° Operazioni	IPFS	MySQL	Fabric
10	0,19	0,19	1,21
100	0,64	0,53	10,29
1000	5,10	3,92	101,3

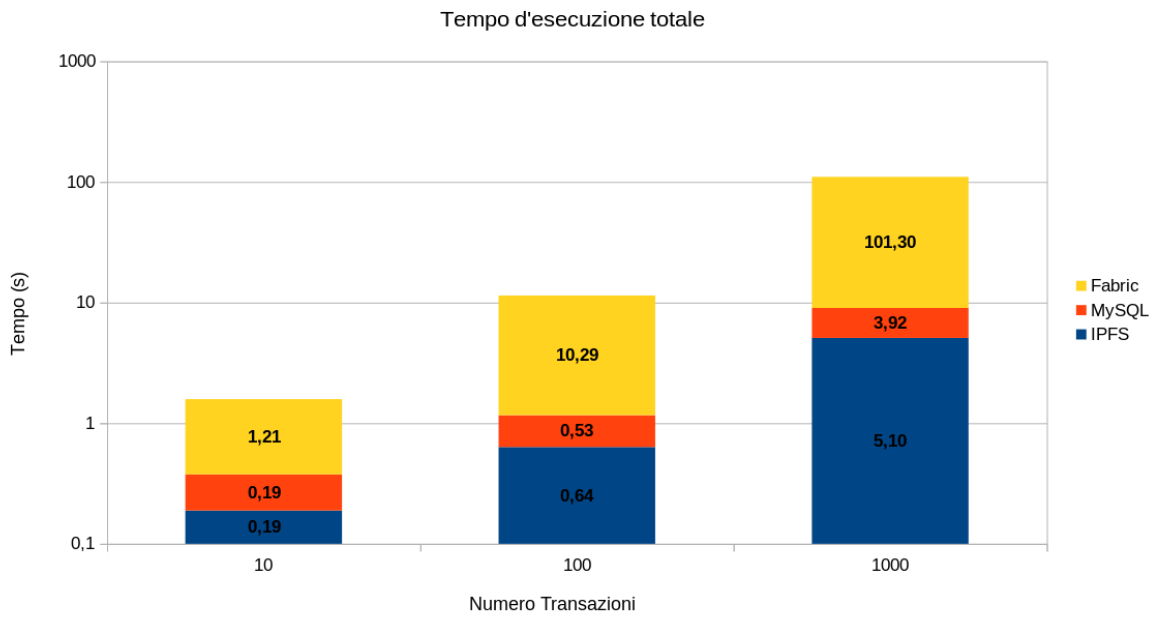

Figura 6.2: Rappresentazione e confronto dei tempi d'esecuzione totale.

Per estrapolare il tempo d'esecuzione totale è stato utilizzato il comando *time* della BASH (Bourne Again SHell) mentre, per la raccolta dei tempi il modulo *time* di Python.

6.3 Secondo test

Per questo test è stato scelto di confrontare due implementazioni diverse di Fabric. La prima implementazione è quella realizzata nel presente elaborato che, come già detto in precedenza, è formata da due canali per la comunicazione in cui ad ogni canale partecipano due organizzazioni, per un totale di tre organizzazioni. Per la seconda implementazione è stata costruita ed eseguita un rete di prova in cui è presente solo un canale, con la partecipazione di tre organizzazioni al suo interno. Come fatto in precedenza, viene definita una prima tabella contenente i dati relativi al tempo d'esecuzione medio di ogni operazione. Nel grafico è possibile notare che il tempo d'esecuzione dell'implementazione con due canali è molto più lento, poiché l'esecuzione di una transazione deve avvenire su due canali (ossia effettua due scritture).

N° Operazioni	Due canali	Un canale
10	105,6	62,89
100	106,4	66,61
1000	98,85	63,39

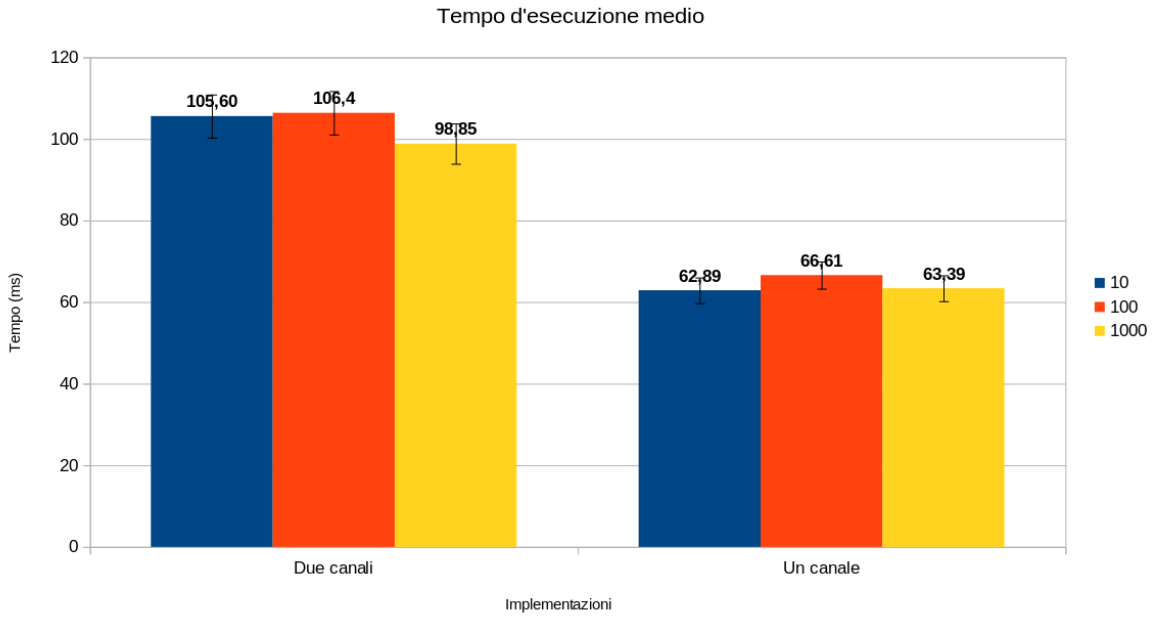

Figura 6.3: Confronto del tempo d'esecuzione medio per ogni operazione.

Nella seconda tabella, i dati rappresentano il tempo d'esecuzione totale che impiega il test a completare tutte le operazioni. Analizzando il grafico in cui i dati sono rappresentati, si nota che l'implementazione con due canali risulta più lenta rispetto a quella con un canale, come già visto in precedenza per il tempo d'esecuzione medio.

N° Operazioni	Due canali	Un canale
10	1,21	0,71
100	10,29	5,66
1000	101,30	56,06

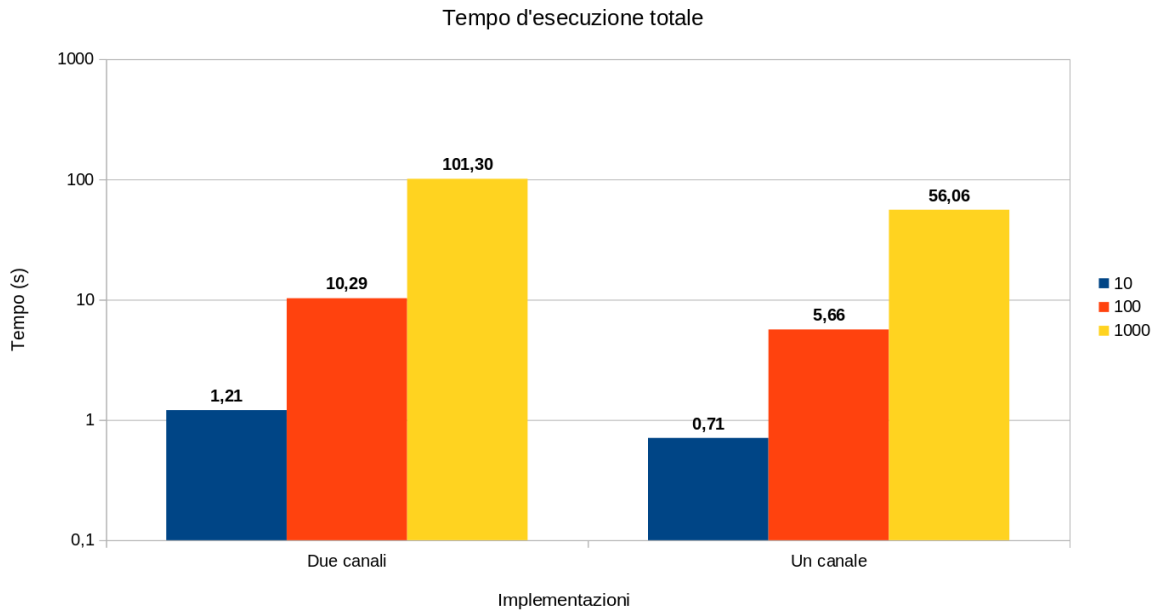

Figura 6.4: Confronto del tempo d'esecuzione totale tra due diverse implementazioni.

CAPITOLO 7

Conclusione e lavori futuri

In conclusione, alla luce di quanto è stato realizzato durante il presente studio, si è constatato che la creazione di un sistema di certificazione digitale basato sulla blockchain permissioned di Hyperledger Fabric risulta un'opzione altamente positiva dal punto di vista della sicurezza e della privacy delle informazioni e delle entità partecipanti al sistema. Riguardo all'aspetto prestazionale, è importante notare che effettivamente i tempi d'esecuzione di qualsiasi operazione si raddoppiano per ogni entità aggiuntiva, poiché ad ogni nuova entità deve essere creato un nuovo canale di comunicazione. Mantenere un alto grado di privatezza e di sicurezza delle informazioni, pur a discapito delle prestazioni, risulta però un vantaggioso compromesso ai fini della realizzazione della presente tesi.
Uno sviluppo futuro potrebbe essere l'aumento della sicurezza tramite l'implementazione di un servizio *IAM* (Identity e Access Management). A tal proposito, sarebbe possibile utilizzare *Keycloak*, il quale oltre alle caratteristiche basilari di un servizio IAM, fornisce altri benefici:

- È una tecnologia completamente open source;

- Può gestire un numero praticamente illimitato di account;
- Risulta molto potente dal punto di vista prestazionale;
- Garantisce delle funzionalità all'avanguardia per la gestione dei dati personali degli utenti.

In breve, questo tipo di servizi mirano a verificare l'identità di un utente per accedere a un sistema o un'insieme di applicazioni, valutando una serie di regole che stabiliscono a quali funzionalità o servizi può accedere un determinato utente.

Bibliografia

«Docker Documentation», https://docs.docker.com/.

«Flask Documentation», https://flask.palletsprojects.com/en/2.0.x/.

«Hyperledger Fabric: A Blockchain Platform for the Enterprise», https://hyperledger-fabric.readthedocs.io/en/release-2.2/.

«InterPlanetary File System Docs», https://docs.ipfs.io/.

«SQLAlchemy documentation», https://docs.sqlalchemy.org/en/14/#.

BUTERIN, V., «The Meaning of Decentralization», https://medium.com/@VitalikButerin/the-meaning-of-decentralization-a0c92b76a274.

RAUCHS, M., GLIDDEN, A., GORDON, B., PIETERS, G., RECANATINI, M., ROSTAND, F. e VAGNEUR, K. (2018), *Distributed Ledger Technology Systems: A Conceptual Framework*, University of Cambridge.

YAGA, D., MELL, P., ROBY, N. e SCARFONE, K. (2018), *Blockchain Technology Overview*, National Institute of Standards and Technology.

Siti Web consultati

- Wiki Hyperledger – `https://wiki.hyperledger.org/`
- Hyperledger – `https://docs.google.com/document/d/1ExFNRx-yYoS8FnDIUX1_0UBMha9TvQkfts2kVnDc4KE/edit#heading=h.rl04ahp2da86`
- Hyperledger Fabric – `https://wiki.hyperledger.org/display/fabric/Design+Documents`
- Medium – `https://medium.com/`
- Dev – `https://dev.to/`
- Red Hat – `https://www.redhat.com/it/topics`
- IBM Cos'è Hyperledger Fabric – `https://www.ibm.com/it-it/topics/hyperledger`